Machine Learning for Dynamic Incentive Problems

Philipp Renner
Department of Economics
University of Lancaster
United Kingdom
p.renner@lancaster.ac.uk

Simon Scheidegger*
Department of Economics
University of Lausanne
Switzerland
simon.scheidegger@unil.ch

January 19, 2025

Abstract

We present a flexible and scalable computational framework that integrates machine learning and optimization theory to efficiently and accurately solve dynamic adverse selection models with persistent private information and many types. Our approach reformulates the model into a numerically tractable structure, bypassing set-valued dynamic programming and making it solvable with standard recursive methods. The recast problem is embedded into a parallelized value function iteration algorithm, where high-dimensional and nonlinear functions are efficiently approximated using Gaussian process regression combined with Bayesian active learning. We apply our framework to two previously intractable models: one with persistent hidden information involving up to ten types and another incorporating multiple persistent types and overreporting. Validation against known solutions and rigorous computational credibility measures confirm the accuracy of our approach.

JEL classification: C61, C73, D82, D86, E61.

Keywords: Dynamic Contracts, Principal-Agent Model, Heterogeneous Agents, Dynamic Programming, Machine Learning, Gaussian Processes.

*We thank Marlon Azinovic, Ben Brooks, Lorenzo Bretscher, Johannes Brumm, Francesco Celentano, Hui Chen, Antoine Didisheim, Rick Evans, Min Fang, Jesús Fernández-Villaverde, Lukas Frank, Luca Gaegauf, Ken Judd, Narayana Kocherlakota, Felix Kubler, Luca Mazzone, John Rust, Karl Schmedders, Tony Smith, Chris Sleet, Aleh Tsyvinski, Yucheng Yang, Sevin Yeltekin, Jan Zemlicka, and seminar participants at Yale University, the University of Lausanne, the University of Zurich, EPFL Lausanne, ETH Zurich, the University of Geneva, MIT Sloan School of Management, Imperial College, CEF 2018 in Milan and LGTC 2018, the NBER Conference on Big Data and High-Performance Computing for Financial Economics 2019, and FinEML 2024 for their valuable comments. This work was generously supported by grants from the Swiss National Supercomputing Centre (CSCS) under project IDs s885, s995, the Swiss Platform for Advanced Scientific Computing (PASC) under project ID "Computing equilibria in heterogeneous agent macro models on contemporary HPC platforms", the Swiss National Science Foundation (SNF), under project IDs "Can Economic Policy Mitigate Climate-Change?", "New methods for asset pricing with frictions", and the Enterprise for Society (E4S). The work was also supported by the High End Computing facility at Lancaster University. Simon Scheidegger gratefully acknowledges support from the MIT Sloan School of Management, the Cowles Foundation at Yale University, and the Department of Economics at the University of Pennsylvania.

1 Introduction

Motivation. Dynamic incentive problems are of key importance in model-based economics, arising whenever contracts are formed under asymmetric information. They feature prominently in applications such as manager remuneration, optimal taxation, and insurance contracts.[2] These problems often involve hidden information about preferences or risk, which complicates the design of optimal contracts. For instance, in insurance markets, adverse selection occurs when insurers cannot differentiate between high- and low-risk individuals, potentially driving low-risk participants out of the market (Akerlof, 1970).

Modeling dynamic incentive problems is particularly challenging because real-world settings often involve persistent private information and heterogeneity across agents. Persistence, such as in health risks or income shocks, contrasts sharply with the common simplifying assumption of independently and identically distributed (i.i.d.) shocks.[3] While persistent shocks better align with empirical data, they introduce substantial complexity, requiring models to track history dependence and to handle irregularly shaped, that is, non-hypercubic, high-dimensional state spaces. To manage these difficulties, researchers often assume more tractable dynamics, such as static contracts (e.g., Sandroni and Squintani, 2007) or a small number of agent types (e.g., Mathevet et al., 2022). These simplifying modeling assumptions affect key economic questions. For instance, classic results suggest long-run immiseration,[4] where most individuals become impoverished over time (Green, 1987, Thomas and Worrall, 1990). Alternative assumptions predict long-run bliss (Williams, 2011). Bridging these extremes, Zhang (2009) demonstrates that intermediate persistence can produce a range of different long-run outcomes. Nevertheless, computational challenges limit the exploration of other cases.

The reliance on simplifying assumptions can create a gap between theoretical models and real-world observations, highlighting the need for more scalable computational methods capable of handling the intricate structure of dynamic incentive problems with persistent private information.

Contribution of this Paper. In this article, we make progress on these issues by proposing a generic and scalable computational framework based on machine learning and optimization theory. This framework enables dynamic adverse selection models with persistent information and many types to be computed accurately and efficiently. Our methodology will enable researchers to solve a wide variety of dynamic incentive problems that were previously considered out of reach. We illustrate the performance of our framework by studying two discrete-time dynamic adverse selection models: one with persistent hidden information involving up to ten types, and the other incorporating

[2] See, e.g., Golosov et al. (2016) for a comprehensive review.

[3] Previous research suggests that private information in the economic environments we are interested in is highly persistent (e.g., Meghir and Pistaferri, 2004, Storesletten et al., 2004).

[4] See Kocherlakota (2010) for a general introduction.

multiple persistent types and limited overreporting;[5] problems that, to the best of our knowledge, have not been studied before despite their inherent relevance.

Challenges in Solving Dynamic Adverse Selection Problems. Two major bottlenecks create substantial difficulties in solving dynamic adverse selection problems with persistent hidden information when existing methods are applied. The first challenge arises from the fact that models with repeated agency require full history dependence (e.g., Lambert, 1983, Rogerson, 1985). This, in turn, leads to time-inconsistent dynamic programs. A common approach to address this issue is to introduce *promised utilities* as artificial state variables (Fernandes and Phelan, 2000). However, this introduces a severe complication in practical applications: the feasible set becomes endogenous to the problem, meaning it must also be computed. Abreu et al. (1986, 1990) provide a constructive proof for the existence of the set of continuation payoffs. Nevertheless, it is generally unclear how to numerically represent possibly multi-dimensional and irregularly shaped sets explicitly.

The second bottleneck relates to solving these models within a "reasonable" computational time. As more agent types are introduced, the dimensionality of the feasible set increases. This set is often irregularly shaped. Consequently, if standard Cartesian grid-based value function iteration algorithms are used in the solution process, the computational effort and storage requirements grow exponentially, rendering even moderately complex models computationally intractable. This effect is commonly referred to as the *curse of dimensionality* (Bellman, 1961).[6] As a result, the existing literature is largely confined to two-dimensional models, where the curse of dimensionality is avoided from the outset (Broer et al., 2017, Doepke and Townsend, 2006, Abraham and Pavoni, 2008).

Proposed Methodology and Preview on Results. In this article, we address these shortcomings through a generic and scalable computational framework. Our contributions are four-fold:

[5]The concept of overreporting refers to an agent's ability to report an income higher than his actual endowment in a dynamic insurance contract. While the literature often assumes, for formal and numerical tractability, that agents cannot overreport their type (e.g., Williams, 2011, Bloedel et al., 2024, and references therein), with the interpretation that agents cannot covertly trade or produce, Bloedel et al. (2020), among others, argue that not incorporating overreporting represents a relaxation of the "full" problem. The latter reflects real-world scenarios where agents can misreport in both directions, allowing for a more comprehensive characterization of incentive-compatible contracts, clarifying the role of tail restrictions (e.g., no-Ponzi constraints), and potentially challenging established results like the failure of immiseration by revealing the suboptimality of certain contracts.

[6]There are two key computational challenges when the solution to a model is obtained by iterating on a Bellman equation. First, in each iteration, one must approximate value and policy functions. Achieving this requires evaluating these functions at numerous points in a high-dimensional state space. Second, at each such point, it is necessary to solve a high-dimensional maximization problem. Together, these features make it difficult to achieve a fast time-to-solution process. To illustrate the intuition behind the curse of dimensionality, consider approximating a univariate value function by evaluating it at 100 points along the state dimension. Extending this to d dimensions requires evaluating the function at $O(100^d)$ points in the d-dimensional state space. Even if individual function evaluations are inexpensive, this naive approach to high-dimensional approximation quickly becomes computationally infeasible.

First, we introduce a reformulation of dynamic incentive problems that is numerically more tractable than the standard recursive approaches commonly used in the literature (e.g., Fernandes and Phelan, 2000, Golosov et al., 2016). Specifically, we employ penalization methods from constrained optimization (Luenberger and Ye, 2008, Chapter 13) to relax the recursive formulation, thereby bypassing the challenging task of set-valued dynamic programming (Abreu et al., 1986, 1990) often required to identify the feasible set. This relaxation allows the problem to be solved with standard value function iteration. To validate our approach, we provide two formal proofs demonstrating that the relaxed formulation recovers the original problem.[7]

Second, to the best of our knowledge, we are the first to solve recursively formulated dynamic incentive problems using a dynamic programming algorithm that combines Gaussian process regression (GPR; e.g., Rasmussen and Williams, 2005) with Bayesian active learning (BAL; e.g., Deisenroth et al., 2009). These machine learning techniques[8] efficiently approximate high-dimensional value and policy functions within value function iteration. BAL plays a critical role in mitigating the curse of dimensionality by concentrating the Gaussian process (GP) approximation on the equilibrium path; a portion of the state space typically much smaller than the full feasible set (e.g., Sannikov, 2022). This approach ensures that, given a fixed computational budget, the highest-quality approximation is achieved where it is most needed. Additionally, we implement a parallelization scheme to leverage high-performance computing, enabling the solution of complex problems with many state variables in a relatively short time.

Third, we validate our framework by applying it to the dynamic adverse selection model of Fernandes and Phelan (2000) with up to ten types, corresponding to a ten-dimensional state space. For their two-dimensional baseline case, where approximate solutions are available, we compare results to assess the accuracy of our method. Since the models we study no longer admit analytical solutions, we incorporate rigorous measures of computational credibility. Following best practices in computational science, specifically *validate, verify, and uncertainty quantification* (VVUQ; Oberkampf and Roy, 2010), we propose four error metrics to jointly assess the correctness of our results.

Fourth, to illustrate the broad applicability of our method, we examine a central question in the contemporary literature: Which assumptions determine the long-run outcomes in adverse selection models? Specifically, we propose a variant with overreporting that allows an agent to claim a higher type without depending on external factors such as hidden borrowing. This intricate set-up introduces additional complementarity constraints, rendering the policy functions non-smooth. Consequently, to address such a model effectively, an advanced algorithm like ours, capable of approximating nonlinear functions in multidimensional settings with irregularly shaped computational domains, is required.

[7] Our relaxation is particularly useful for dynamic adverse selection problems with persistent shocks and more than two types. It also simplifies two-dimensional settings by reducing the solution procedure from computing the feasible set and solving the dynamic model sequentially to standard dynamic programming.

[8] Appendix B provides a short glossary of terms that we use in this paper and that are common in the machine learning literature. In addition, we try to relate the machine learning terminology to the terms commonly used in economics.

Key findings from the two- and four-type versions of the overreporting model reveal that allowing overreporting markedly alters the distribution of long-run outcomes. Consumption tends to concentrate away from extreme values, and the agent's utility promises become more heavily smoothed over time, in contrast to the baseline model's outcomes that features no overreporting. This difference arises only when combining overreporting with multiple types, underscoring the significant role both assumptions play in long-run contract behavior.

Finally, we provide complementary code examples illustrating our methodology, available at https://github.com/GaussianProcessesForDynamicEcon/DynamicIncentiveProblems.

Organization of Paper. The remainder of this article is organized as follows. In Section 2, we provide a brief review of the related literature. Section 3 presents a standalone introduction to GPs and BAL accompanied by various numerical experiments to illustrate their mechanics. In Section 4, we outline our generic solution framework in a cookbook-style format, apply it to a benchmark model, and provide the theoretical proofs that underpin our method. We also discuss the computational advantages of our framework relative to existing methods. Section 5 extends the applicability of our approach by solving a second type of model: a dynamic adverse selection model with overreporting. Finally, Section 6 concludes.

2 Related Literature

Previous research has extensively studied various approaches to make dynamic incentive models formally tractable.[9] However, the numerical treatment of these models has often been tailored to specific instances, leaving many problem formulations intractable. This restriction in the number of state variables that can be efficiently handled is particularly problematic, as additional state variables are frequently needed to address the questions of interest. Kocherlakota (2005), when solving an optimal taxation model, states: *"It would be desirable to use an infinite horizon example as in Albanesi and Sleet (2006). However, to answer the questions of interest, the example would have to include aggregate shocks, persistent hidden state variables, and probably should allow for endogenous physical state variables. At a conceptual level, it is known how to attack problems of this kind, thanks to the work of Fernandes and Phelan (2000) and Doepke and Townsend (2006). Practically, it is still impossible to implement their procedures in an example that includes the elements of interest."* The work presented in this paper aims to complement these formal approaches by providing a generic and scalable computational framework that renders such problems computationally tractable.

[9]For an incomplete list of research in discrete-time settings, see, e.g., Spear and Srivastava (1987), Fernandes and Phelan (2000), Cole and Kocherlakota (2001), Werning (2002), Doepke and Townsend (2006), Abraham and Pavoni (2008), Mele (2014), Pavoni et al. (2017), Kapička (2013), Pavan et al. (2014), and for models in continuous time, see, e.g., DeMarzo and Sannikov (2006), Sannikov (2008), Williams (2009, 2011), He et al. (2017).

The seminal work by Abreu et al. (1986, 1990) (henceforth APS) introduced a constructive procedure for computing feasible sets. Specifically, the authors demonstrated the existence of a monotone set-valued operator whose fixed point corresponds to the feasible set, akin to the Bellman operator in dynamic programming. In practical applications, solving such models recursively requires repeatedly approximating non-convex equilibrium correspondences using numerical techniques. For instance, Judd et al. (2003) and Yeltekin et al. (2017) propose a numerical scheme for determining feasible sets in discrete-state supergames using polygons. However, their approach relies on convexifying the payoff set and suffers from the curse of dimensionality. Similarly, Sleet and Yeltekin (2016) extend this method to continuous state variables, but it faces the same limitations. Methods proposed by Abreu and Sannikov (2014) for discrete-state, two-player games are restricted to convex sets and also subject to the curse of dimensionality. Abreu et al. (2020) improve convergence by distinguishing between bounding and slack incentive constraints, but their polytope-based approach is limited to specific high-dimensional cases. In contrast, our proposed approach addresses these challenges with several desirable features. First, by reformulating dynamic incentive problems to introduce slack variables in infeasible regions of the computational domain, we eliminate the need to pre-determine the feasible set, thus transforming the problem into a standard dynamic programming task (Stokey et al., 1989). Second, our computational framework alleviates the curse of dimensionality, enabling it to address problems with many types and additional state variables, if necessary. Third, it is flexible enough to accommodate both convex and non-convex feasible sets. Additionally, while pre-computation of the feasible set is unnecessary, it can still be approximated using information implicitly available from the model solution.

Several authors have proposed alternatives to the approach by Fernandes and Phelan (2000) for addressing dynamic incentive problems. For instance, Marcet and Marimon (2019) study a planner's problem with forward-looking constraints, introducing Lagrange multipliers as state variables. This leads to a recursive saddle point formulation, and avoids the need to determine a feasible set. Pavoni et al. (2017) extend this approach to more general constraints, restricting the state space to the positive orthant, while Mele (2014) further expands the method to include hidden actions. However, there are two major drawbacks to using the Lagrangian approach. First, it requires strict assumptions regarding the curvature of the functions involved; specifically, the convexity of the planner's problem. Second, the state space in all recursive formulations proposed by Mele (2014), Pavoni et al. (2017), and Marcet and Marimon (2019), is non-compact, which renders standard numerical dynamic programming techniques (e.g., Judd, 1998) inapplicable, as these methods require a compact domain (Stokey et al., 1989). Alternatively, Kapička (2013) and Pavan et al. (2014) examine a continuum of types using a first-order approach to solve a relaxed problem. This method involves tracking utility promises and marginal utility alongside physical state variables. However, their approaches impose strong assumptions on the distribution of hidden information and require ex-post verification of results. As with the case of finitely many types, the feasible set remains unknown a priori, necessitating its determination beforehand. To simplify the first-order approach for

numerical applications, Kapička (2013) introduces additional assumptions to reduce the model to two dimensions, whereas Pavan et al. (2014) pursue a purely analytical approach. Despite these efforts, the first-order approach is unsuitable for multi-dimensional heterogeneity due to stringent assumptions on the information distribution. For example, while it accommodates continuous income shocks, it cannot handle additional heterogeneity, such as different risk types, without violating these assumptions. In contrast, discrete-state models allow for greater flexibility in incorporating heterogeneity by simply adding more types, albeit at the cost of increased dimensionality. Moreover, Battaglini and Lamba (2019) demonstrate that the first-order approach is generically invalid, meaning it fails for most assumptions about preferences and distributions. Even with a single continuous type, such models exhibit irregularly shaped state spaces of at least three dimensions. These issues highlight the potential benefits of our complementary computational approach, which can accommodate such settings if reformulated appropriately.

Once a dynamic incentive problem is reformulated, as proposed in this article, computing global solutions becomes necessary.[10] To achieve this, we introduce a parallelized discrete-time dynamic programming algorithm that uses GPR to approximate value and policy functions and employs distributed memory parallelism (e.g., Skjellum et al., 1999) to distribute the computational workload. Additionally, we enhance the efficiency of GPR by applying BAL (e.g., Krause and Guestrin, 2007) to focus data collection on regions where precision is most needed. This combination enables highly accurate function approximations with minimal observations. GPR, a supervised machine learning technique, has been widely applied in data science, engineering, and other fields for function approximation and classification tasks. Early applications include Engel et al. (2003), who used GPs in reinforcement learning to navigate a two-dimensional maze, and Deisenroth et al. (2009), who employed GPs and BAL for approximating value and policy functions in low-dimensional control problems, such as pendulum swings. Berkenkamp et al. (2016) utilized GPs for safe controller optimization in quadrotors, and Busoniu et al. (2010) provide a textbook treatment of reinforcement learning with kernel-based function approximators. In economics, Scheidegger and Bilionis (2019) applied GPs to solve dynamic stochastic growth models and conduct uncertainty quantification, while Kotlikoff et al. (2021) integrated aspects of this methodology into a time iteration algorithm (Chapter 17.8, Judd, 1998) to analyze overlapping generation models in the context of climate change, and Gaegauf et al. (2023) to investigate dynamic portfolio choice problems with transaction costs. We extend this prior work by combining GPs with BAL to efficiently mitigate the curse of dimensionality and solve dynamic adverse selection models featuring persistent information and many types.

A defining feature of GPs is their ability to *learn*—approximating functions in a non-parametric manner using observations at so-called design points, without geometric re-

[10] Following Brumm and Scheidegger (2017), we use the term "global solution" to refer to a solution computed using equilibrium conditions at multiple points in the state space of a dynamic model, in contrast to a "local solution", which relies on a local approximation around a steady state, as is common in perturbation methods. We refer to methods that compute such global solutions as "global solution methods." This use of "global" should not be confused with its use in the term "global optimization method."

strictions. This contrasts sharply with grid-based approximation schemes commonly used for high-dimensional state spaces, such as Smolyak's method (e.g., Krueger and Kubler, 2004, Judd et al., 2014, Fernández-Villaverde et al., 2015), adaptive sparse grids (e.g., Brumm and Scheidegger, 2017, Brumm et al., 2022), and high-dimensional model representation (e.g., Eftekhari and Scheidegger, 2022). These grid-based methods are typically constrained to hyper-rectangular state spaces, making them less suitable for solving dynamic adverse selection models, where state spaces are often irregularly shaped.

The long-run ergodic distribution is a central topic in dynamic adverse selection models. Several studies, including Green (1987) and Thomas and Worrall (1990), find that long-run contracts can lead to immiseration.[11] This extreme outcome raises the question: Which assumptions drive it? Traditionally, dynamic incentive models prevent agents from overreporting their type, often by invoking a no hidden borrowing constraint (e.g., Kocherlakota, 2010, Bloedel et al., 2020, and references therein), supported by empirical evidence (see, e.g., Feldman and Slemrod, 2007, from US tax data), but also for tractability. Yet, overreporting can be crucial in certain adverse selection problems, enabling misreporting strategies such as overstating assets to secure loans or exaggerating performance to attract investors. It may also play a key role in understanding polarizing long-run outcomes like "immiseration" or "bliss". We contribute to this literature by introducing limited overreporting, which permits agents to overreport only when external intervention, such as hidden borrowing, is not required. Concretely, we modify the incentive compatibility constraint so that it switches off whenever overreporting would require borrowing, which introduces kinks in policy functions and poses a substantial numerical challenge, especially in high-dimensional settings. Our proposed method can help bridge this gap. Because it handles richer model environments than previously considered, it enables researchers to remove the "straitjacket" imposed by the present computational restrictions.

3 An Introduction to Gaussian Process Regression

This section proceeds in three main steps. First, in Section 3.1 we briefly introduce GPs and how they can be used to approximate and interpolate multi-dimensional functions. Second, Section 3.2 introduces BAL, a method from reinforcement machine learning that can strategically determine the locations in the state space where functions have to be evaluated to maximize the quality of the GP function approximation with as few points as possible. Third, in Section 3.3, we provide simple numerical examples that illustrate the joint workings of GPs and BAL.

[11] For a recent discussion, see Bloedel et al. (2024).

3.1 Function Approximation with Gaussian Processes

In the following, we introduce GPR, a nonparametric regression method from supervised machine learning that has universal function approximation properties (e.g., Micchelli et al. (2006)) and that we will use to approximate and interpolate multivariate policy and value functions (e.g., Rasmussen and Williams, 2005, and Murphy, 2012).

Given a so-called *training data* set $\mathcal{D} = \{(x_i, y_i)|i = 1, \ldots, N\}$, consisting of N input states $x_i \subset \mathbb{R}^d$ and corresponding observations $y_i \in \mathbb{R}$,[12] we model the data generating process as $y_i = f(x_i) + \varepsilon$, where $f : \mathbb{R}^d \to \mathbb{R}$ is an underlying function of interest, and $\varepsilon \sim \mathcal{N}(0, \sigma_\epsilon^2)$ is a noise term.[13] In the literature, $\mathcal{X} = \{x_1, \ldots, x_N\}$ is often referred to as *training inputs*, whereas $\mathcal{Y} = \{y_1, \ldots, y_N\}$ contains the corresponding observations, and is called *training targets*. The latter can, as in our case, be generated via computer code,[14] or stem from empirical data. Subsequently, we will often refer to the stacked training data as training inputs $\mathbf{X} \in \mathbb{R}^{N \times d}$ and targets $\mathbf{y} \in \mathbb{R}^N$.

To enable predictions based on information contained in $\mathcal{D}$, we must make assumptions about the characteristics of the underlying functions. A GP defines a distribution over functions, such that if we pick any finite set of inputs, the corresponding stochastic outputs (i.e., the evaluations of the functions from that distribution) follow a joint Gaussian distribution.

More formally, let $\mu : \mathbb{R}^d \to \mathbb{R}$ and $k : \mathbb{R}^d \times \mathbb{R}^d \to \mathbb{R}$ be the *mean function* and *covariance function*, respectively. We say that a stochastic function $f : \mathbb{R}^d \to \mathbb{R}$ is distributed as a GP with mean function μ and covariance function k, denoted $f \sim \mathcal{GP}(\mu, k)$. For any $N \in \mathbb{N}$ and any set of inputs $\{x_i\}_{i=1,\ldots,N}$, the outputs are jointly distributed according to an N-variate normal distribution:

$$[f(x_i)]_{i=1,\ldots,N} \sim \mathcal{N}_N \left([\mu(x_i)]_{i=1,\ldots,N}, \left[k(x_i, x_j)\right]_{i,j=1,\ldots,N} \right), \tag{1}$$

where the bracket notation $[\mu(x_i)]_{i=1,\ldots,N}$ denotes stacking all N real-valued elements $\mu(x_i)$ into a vector, and similarly, $[k(x_i, x_j)]_{i,j=1,\ldots,N}$ into an $N \times N$ matrix.

[12] For notational simplicity, we restrict ourselves to the univariate output case. However, all expressions derived in the following carry over to the situation where GPs have to deal with multivariate output (Bilionis et al., 2013).

[13] This assumption is similar to that made in linear regression in that we assume an observation consists of an independent "signal" term $f(x)$ and a "noise" term ϵ. In GPR, however, we assume that the signal term is also a random variable that follows a particular distribution. This distribution is subjective in the sense that it reflects our uncertainty regarding the function. The uncertainty regarding f can be reduced by observing the output of the function at different input points. The noise term ϵ reflects the inherent randomness in the observations, which is always present no matter how many observations we make. In empirical setups, measurement noise may arise from our inability to control all the influential factors or irreducible (aleatory) uncertainties. In computer simulations, measurement uncertainty may arise from limitations in algorithmic accuracy or hardware constraints in representing floating-point numbers, though it is generally small.

[14] In our concrete case below, the training data will be obtained by solving a constrained optimization problem as stated for instance in Eq. (27) at N strategically chosen points x_i from within a feasible set (see, e.g., Section 4.5 for more details), and the corresponding observations y_i are given by the value and policy functions at that particular location in the state space, respectively.

The mean function $\mu(x)$ reflects the expected function value at input x:

$$\mu(x) = \mathbb{E}[f(x)], \tag{2}$$

that is, the average of all functions in the distribution evaluated at input x. The prior mean function is required to model any general trends of the response surface and can have any functional form. However, it is often set to $\mu(x) = 0$ to simplify posterior computations, allowing inference solely via the covariance function. For simplicity, we will set it to 0 for the remainder of this section. The covariance function $k(x, x')$ models the dependence between the function values at different input points x and x':

$$k(x, x') = \mathbb{E}\left[(f(x) - \mu(x))\left(f(x') - \mu(x')\right)\right]. \tag{3}$$

The function k is commonly called the *covariance kernel* of the GP (Rasmussen and Williams, 2005). The choice of an appropriate kernel is the most crucial part of GPR and needs to be based on assumptions such as smoothness and likely patterns to be expected in the data. A sensible assumption is usually that the correlation between two points decays with the distance between the points. This means that closer points are expected to behave more similarly than points that are further away from each other. One very popular choice of a kernel fulfilling this assumption is the radial basis function kernel (also known as the *square exponential (SE) kernel*), which is defined as

$$k_{\text{SE}}(x, x'; \phi) = s^2 \exp\left\{-\frac{1}{2}\sum_{j=1}^{d}\frac{\left(x^j - x'^j\right)^2}{\ell_j^2}\right\}, \tag{4}$$

with *hyperparameters* $\phi = \{s, \ell_1, \ldots, \ell_d\}$, with $s > 0$ being the variability of the latent function f, and $\ell_j > 0$ the characteristic lengthscale of the j-th input dimension.[15] The exact choice of the kernel within an application boils down to how the modeler encodes prior knowledge about the function(s) to be approximated, such as differentiability and periodicity. In our applications below, we typically work with SE or piecewise polynomial kernels (for more details on kernels, see, e.g., Murphy (2022, Chapter 17), and https://www.cs.toronto.edu/~duvenaud/cookbook).

Suppose we have collected observations $\mathcal{D} = \{\mathbf{X}, \mathbf{y}\}$, for example, by solving N Bellman equations at various locations x_i, and we want to make predictions for new inputs, collected in the matrix $\mathbf{X}_*$, by drawing the corresponding $\mathbf{f}_*$ from the posterior distribution $p(f \mid \mathcal{D})$. According to the definition of the GP, the previous observations $\mathbf{y}$ and function values $\mathbf{f}_*$ follow a joint multivariate normal distribution, which can also be written as

$$\begin{bmatrix} \mathbf{y} \\ \mathbf{f}_* \end{bmatrix} \sim \mathcal{N}\left(\mathbf{0}, \begin{bmatrix} k(\mathbf{X}, \mathbf{X}) + \sigma_\epsilon^2 \mathbf{I} & k(\mathbf{X}, \mathbf{X}_*) \\ k(\mathbf{X}_*, \mathbf{X}) & k(\mathbf{X}_*, \mathbf{X}_*) \end{bmatrix}\right), \tag{5}$$

[15] Note that the hyperparameters of the covariance function are typically estimated by maximizing the likelihood (Rasmussen and Williams, 2005).

where $k(\mathbf{X}, \mathbf{X})$ is the covariance matrix between all observed points so far, $k(\mathbf{X}_*, \mathbf{X}_*)$ is the covariance matrix between the newly introduced set of points for which we want to make a prediction, $k(\mathbf{X}_*, \mathbf{X})$ is the covariance matrix between the new input points and the already observed points, and $k(\mathbf{X}, \mathbf{X}_*)$ is the covariance matrix between the observed points and the new input points. Moreover, $\mathbf{I}$ is an identity matrix, and σ_ϵ^2 is the assumed noise level of observations, that is, the variance of ϵ. Using standard results (e.g., Rasmussen and Williams, 2005 and Rasmussen and Nickisch, 2010), the conditional distribution $p(\mathbf{f}_* \mid \mathbf{X}, \mathbf{y}, \mathbf{X}_*)$ is then a multivariate normal distribution with a posterior, which by itself is a GP with mean function

$$\tilde{\mu}(x) = k(x, \mathbf{X}) \left[k(\mathbf{X}, \mathbf{X}) + \sigma_\epsilon^2 \mathbf{I}\right]^{-1} \mathbf{y}, \tag{6}$$

and a covariance kernel that reads:

$$\tilde{k}(x, x') = k(x, x') - k(x, \mathbf{X}) \left[k(\mathbf{X}, \mathbf{X}) + \sigma_\epsilon^2 \mathbf{I}\right]^{-1} k(\mathbf{X}, x'). \tag{7}$$

The previous two equations imply that calculating the posterior mean and covariance of a GP involves first calculating the four different covariance matrices contained in Eq. (5) and then combining them according to Eqs. (6) and (7). Inverting the matrix via Cholesky decomposition, this step scales as $O(N^3)$ with the number of observations N in the training set $\mathcal{D}$ (Rasmussen and Williams, 2005).[16] In order to predict $\mathbf{f}_*$, we can simply use the mean function, our best estimate for the desired value, which is given by Eq. (6), and evaluate it at the location of interest: $\tilde{\mu}(x_*)$. Furthermore, Eq. (7) provides the *predictive variance* $\tilde{\sigma}^2(x_*) := \tilde{k}(x_*, x_*)$. The latter can be used to derive point-wise predictive error bars and thus provide information about the model confidence, that is, about the quality of the function approximator at a point x_*.

Note that the predictive mean $\tilde{\mu}(x)$ given by Eq. (6) can also be written as

$$\tilde{\mu}(x) = \sum_{i=1}^{N} a_i k(x_i, x), \tag{8}$$

where each x_i is a previously observed input value in $\mathbf{X}$, and the weights are collected in the vector $\boldsymbol{a}^T = (a_1, \ldots, a_N) = \left(k(\mathbf{X}, \mathbf{X}) + \sigma_\epsilon^2 \mathbf{I}\right)^{-1} \mathbf{y}$. Intuitively, we can think of the GP posterior mean as an approximation of f using N symmetric radial basis functions (RBFs) centered at each observed input. Thus, by choosing a covariance function $k(x, x')$ that vanishes when x and x' are separated by a lot, for example, the SE covariance function (cf. Eq. (4)), we see that an observed input-output will only affect the approximation locally. This observation also establishes a connection between GPR and reproducing-kernel Hilbert spaces, which are discussed in Rasmussen and Williams (2005). Moreover, the radial basis

[16] Recent advances in algorithmic design have substantially reduced this bottleneck. For instance, the method proposed by Gardner et al. (2018), which we employ in our numerical experiments below, decreases the asymptotic complexity of exact GP inference from $O(N^3)$ to $O(N^2)$. For more efficient methods to address this performance bottleneck, see Appendix C.

function has universal approximation and regularization capabilities. Theoretically, the RBF can approximate any continuous function arbitrarily well (e.g., Poggio and Girosi, 1990, Park and Sandberg, 1991, and Wu et al., 2012). For more details on why, in the age of deep learning, GPs can (and should) be used in the presented context of models, see Appendix E, and references therein.

3.2 Exploring the State Space by Bayesian Active Learning

To efficiently approximate functions with GPs, we need to identify a set of states $\mathbf{X}$ that sufficiently covers the relevant state space, enabling the GP to approximate, for instance, value and policy functions accurately with a minimal amount of training data $\mathcal{D}$. The challenge is to determine which data to gather to learn the function(s) of interest as quickly as possible, especially in the case of value function iteration (cf. Sections 4.2.3 and 4.4), where training data is expensive to acquire due to the need to solve many constrained optimization problems in each step of the solution process.

The simplest way to generate training data is to place the observation points so as to train the Gaussian processes uniformly from some compact set $[a, b] \subset \mathbb{R}^d$, $a, b \in \mathbb{R}^d$, for example, via a Halton sequence (Halton, 1964 and Niederreiter, 1992). This strategy can be highly inefficient, particularly in situations where functions that have to be approximated show distinct local features, such as varying gradients, since the observations that are used to generate a good approximation may not necessarily be created where they improve on the approximation quality most. To ensure a good approximation by naive sampling from the domain of interest, one would have to substantially increase the number of observations, as newly added training points are not guaranteed to be added to the training set where they could most improve the quality of the function approximation. Since the training of standard GPs scales cubically with the number of observations N, that is, $\mathcal{O}(N^3)$, and even the method we use in our practical applications scales as $\mathcal{O}(N^2)$, the runtime increases drastically with increasing sample size (cf. Appendix C for more details).

To improve matters substantially, we propose to use Bayesian Active Learning (e.g., MacKay, 1992, Chaloner and Verdinelli, 1995, Krause et al., 2008, Deisenroth et al., 2009, and Makarova et al., 2022). BAL, a reinforcement learning technique, autonomously selects observations in regions where they most improve the approximator's quality based on a chosen metric.[17]

In the following, we formally introduce BAL using GPs before providing concrete examples in Section 3.3. Given a *training data* set $\mathcal{D} = \{\mathbf{X}, \mathbf{y}\}$ consisting of N input states $x_i \in \mathcal{F} \subset \mathbb{R}^d$ and corresponding observations $y_i \in \mathbb{R}$, we aim to select new data points iteratively from a set of unlabeled data points $\mathcal{U} = \{\tilde{x}_j\}_{j=1}^{s}$ to improve the model, that is, the quality of the function approximation.[18] The process of BAL with GPs involves the

[17] In loose terms, BAL could be considered to be the grid-free equivalent of an adaptive sparse grid algorithm (Brumm and Scheidegger, 2017).

[18] Unlabeled data in machine learning consist of input examples without corresponding target labels or

following steps:

1. **Model Update:** Given the current labeled dataset $\mathcal{D}$, fit a GP. This involves updating the posterior distribution of the function f based on the training data:

$$f \mid \mathcal{D} \sim \mathcal{GP}(\tilde{\mu}, \tilde{k}). \tag{9}$$

   where $\tilde{\mu}$ and $\tilde{k}$ are given by Eqs. (6) and (7).

2. **Acquisition Function:** For each unlabeled data point $\tilde{x}_j \in \mathcal{U}$, we need to assess the potential benefit of incorporating an observation at this location into our GP model. To do so, we have to define a so-called *score* or *acquisition function* $\alpha(\tilde{x}_j; \mathcal{D})$. A popular and intuitive example of such a function, which we utilize in our numerical examples in Section 3.3, is defined by the following expression (Deisenroth et al., 2009):

$$\alpha(\tilde{x}_j; \mathcal{D}) = \sigma_m \tilde{\mu}(\tilde{x}_j) + \frac{\sigma_v}{2} \log\left(\tilde{\sigma}(\tilde{x}_j)\right), \tag{10}$$

   where σ_m and σ_v are positive weighting factors, and where $\tilde{\mu}$ and $\tilde{\sigma}$ are the predictive mean and variance of a GP, trained over $\mathcal{D}$, and evaluated at a point $\tilde{x}_j$ that is not contained in the training set, respectively. The first term of Eq. (10), the predictive mean, expresses how much total reward is expected from $\tilde{x}_j$. The second term of Eq. (10), the predictive variance, measures how uncertain the GP approximation is expected to be, given $\mathbf{X}$, in terms of Shannon entropy (Chaloner and Verdinelli, 1995). The parameters σ_m and σ_v control exploitation and exploration, respectively, and must be set manually. Thus, for a given observation $\tilde{x}_j$, one can evaluate Eq. (10) and assign a score on how important it would be to include this particular observation in the training set.[19]

3. **Data Selection:** Select the data point $\tilde{x}^* = \arg\max_{\tilde{x}_j \in \mathcal{U}} \alpha(\tilde{x}_j; \mathcal{D})$ and acquire its label $\tilde{y}^*$, that is, the function value at the given point.[20]

4. **Dataset Update:** Update the labeled dataset $\mathcal{D} \leftarrow \mathcal{D} \cup \{(\tilde{x}^*, \tilde{y}^*)\}$.

These four steps are repeated iteratively until a stopping criterion is met, such as achieving a desired model performance or reaching a maximum number of labeled sample points.[21]

---

outputs. It is primarily used in unsupervised learning to discover patterns, structures, or relationships within the data without predefined categories. For more details, see Appendix B.

[19] In BAL, a wide variety of gradient-free and gradient-based (Wu et al., 2017) acquisition functions is available (e.g., Balandat et al., 2020 and Di Fiore et al., 2024, for summaries). The specific choice of acquisition functions in practice typically depends on the model and often requires numerical experimentation. A comprehensive discussion of this topic, however, lies beyond the scope of the present paper.

[20] In the context of value function iteration, acquiring $\tilde{y}^*$ translates to solving a constrained optimization problem, for instance, Eq. (27).

[21] To construct a training set $\mathcal{D}$ in practice that leverages BAL (cf. Section 4.5), we follow the literature and proceed as follows: First, we start from a small set $\mathbf{X}$ of initial input locations that are generated via a Halton sequence from a set $\mathcal{F}$ and train a GP on the resulting training targets $\mathbf{y}$. Second, we generate s candidate points $\tilde{x}_j \in \mathcal{F}$ via simulations at which we evaluate Eq. (10) given the fitted GP at every $\tilde{x}_j$. Third, we rank

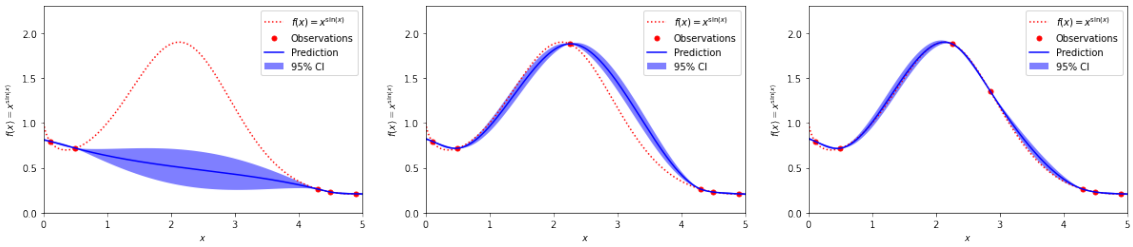

Figure 1: The left panel shows a training data set $\mathcal{D} = \{\mathbf{X}, \mathbf{y}\}$ (red dots) that was generated by evaluating the analytical function $y = x^{\sin(x)}$ (red dashed line) at the points $\mathbf{X} = \{0.1, 0.5, 4.3, 4.5, 4.9\}$ to which we fitted a GP. Furthermore, the predictive mean as well as the 95 percent confidence intervals, corresponding to the point-wise predictive mean (blue line) plus and minus two standard deviations (blue shaded area) for each input value of predictive variance, are also shown. The figure in the center shows the same, but with one observation strategically inserted into the training set around $x = 2.2$ via BAL. The panel on the right enhances the training set by another data point around $x = 2.9$ that was added via BAL.

3.3 Analytical Examples

We now demonstrate the ability of the joint workings of GPs and BAL to efficiently approximate and interpolate functions on two analytical examples $f : [a, b] \to \mathbb{R}, [a, b] \subset \mathbb{R}^d, a, b \in \mathbb{R}^d$, which are adopted from a representative suite of test problems by Genz (1984).[22] We generate various GP approximations to those functions, and measure their accuracy as follows: We randomly generate $N_{test} = 1,000$ test points x_i from a uniform distribution on $[a, b] \subset \mathbb{R}^d$, $a, b \in \mathbb{R}^d$, and compute the mean approximation error, which is given by

$$\frac{1}{N} \sum_{i=1}^{N} |f(x_i) - \tilde{\mu}(x_i)|, \tag{11}$$

where $\tilde{\mu}(x_i)$ is the predictive mean of the GP approximating the test function f.

As a first example, we consider $y = f(x) = x^{\sin(x)}$, where $x \in [0, 5]$. The initial training set $\mathcal{D} = \{\mathbf{X}, \mathbf{y}\}$ consists of only five sample points, to which we fit a GP (cf. the left panel of Fig. 1). Next, we generate $N_{cand} = 1,000$ candidate points and score them according to Eq. (10). The top-ranked sample is added to the training set, and the GP approximation is re-computed (cf. the middle panel of Fig. 1). This process is repeated one more

all the observations $\tilde{x}_j$ based on their score (e.g., Eq. (10)) and add the point with the highest score to our set of training inputs. Thus, the set $\mathbf{X}$ dynamically grows as long as the size of the training set is below some pre-defined size, but only in the regions of the state space where the observations matter, that is to say, in a data-efficient fashion.

[22]The corresponding Python codes illustrating BAL can be found under the following URL: https://github.com/GaussianProcessesForDynamicEcon/DynamicIncentiveProblems/tree/main/analytical_examples.

time (cf. the right panel of Fig. 1). By adding only two extra sample points, the mean approximation error drops from 51% down to 1%. From those three panels it becomes clear that using GPs in combination with BAL can very rapidly increase the quality of the function approximation by monitoring where one is uncertain about the predictive mean and improving on it by adding observations at locations of the computational domain where they are needed most.

In the second example, we consider a two-dimensional test from the "Gaussian peak family" of functions, that is,

$$f(x) = \exp\left(-\sum_{i=1}^{d} a_i^2 (x_i - u_i)^2\right), \tag{12}$$

and choose $d = 2$, $x \in [0,1]^2$. Furthermore, we set $a_i = (5,5)$ and $u_i = (0.8, 0.8)$. The top left panel of Fig. 2 depicts the resulting function. We have chosen this test case as it loosely resembles the shape of the value functions we wish to approximate below in our dynamic incentive models. To approximate the function (12), we proceed as follows: We start by generating an initial training set consisting of 100 points randomly drawn from the two-dimensional domain. Next, we enhance the training set via BAL by an additional 40 points and compute the corresponding approximation errors as a function of the sample size. We compare this setup to the case where the training set is naively enhanced by adding sample points randomly from the domain of interest. The top right panel of Fig. 2 displays the extra points added to the training set via BAL. We depict them in two individual batches of the first and second 20 points. This figure illustrates that BAL adds new sample points to the training set where they are needed most to improve the performance of the approximator, that is, in the top right corner and at the boundary of the domain. In addition, we display the extra 40 uniform sample points we generated, indicating that their random placement is not optimal for increasing the performance of the GP. The lower left panel of Fig. 2 compares the mean approximation error for the GPs, generated from the BAL-enhanced as well as the naively enhanced training set. Strikingly, BAL outperforms the naive approach by about one order of magnitude. Finally, the right panel of Fig. 2 compares the mean approximation error for the GPs to adaptive sparse grids, a state-of-the-art method to efficiently approximate multi-variate functions (Brumm and Scheidegger, 2017). The data points reported for the adaptive sparse grid were obtained by computing the approximation error at grid levels 3 to 7 and a refinement threshold $\epsilon = 10^{-6}$. The figure shows that the joint working GPs and BAL can save substantial resources and outperform advanced methods such as adaptive sparse grids.

4 Solving High-Dimensional Dynamic Incentive Problems

In this section, we systematically introduce a generic solution method based on GPs and BAL (as described in Section 3), designed to solve dynamic incentive models of

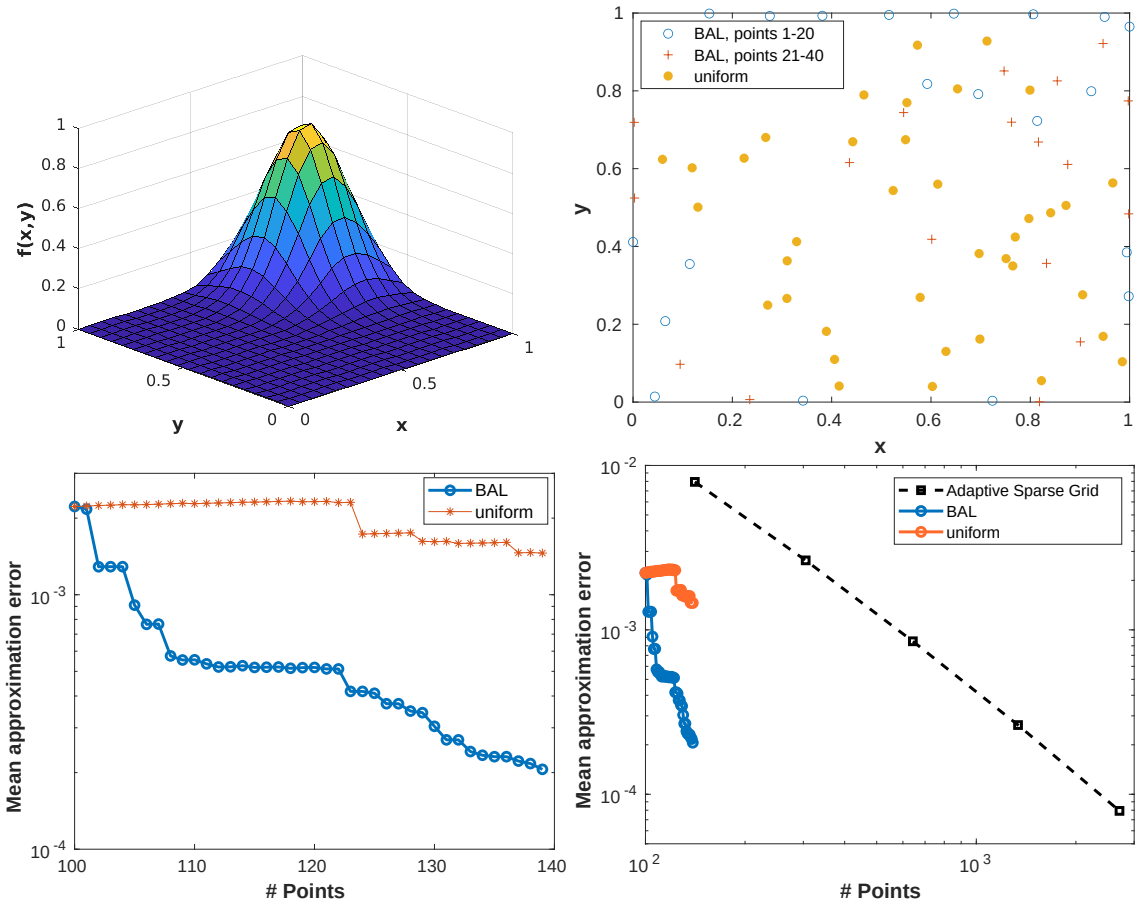

Figure 2: The top left panel depicts the test function (12). The top right panel displays how BAL successively enhances the training set by 40 points. The first 20 points added to this set are depicted as (blue) dots, whereas the subsequently added 20 points are labeled as (red) crosses. This situation is contrasted by adding uniform 40 sample points to the original training set (labeled as yellow points). The lower left figure depicts the interpolation error (see Eq. (11)) of GPs approximating f naively (denoted as "uniform"), or via BAL (denoted as "BAL"). The lower left panel shows the same and compares the performance to an adaptive sparse grid of increasing size.

unprecedented complexity. We illustrate its capabilities with a risk-averse agent who faces unobserved, persistent taste shocks and a risk-neutral planner providing optimal, incentive-compatible insurance. In Section 4.1, we present a baseline model. Section 4.2 introduces a generic, numerically tractable penalty reformulation of dynamic adverse selection problems. As we will prove with two lemmas, this reformulation has the advantage that computing solutions to the reformulated problem comes down to solving an ordinary dynamic programming problem via value function iteration. The procedure for numerically validating our solution method is detailed in Section 4.3. Section 4.4 presents our value function iteration algorithm, which leverages GPs and BAL to solve

the model recursively. Finally, in Section 4.5, we evaluate our method's performance for the baseline model with up to ten types (i.e., ten dimensions). Because the dimensionality of this dynamic incentive model grows linearly with the number of shocks considered, it is ideally suited for demonstrating our approach's computational efficiency. In addition, Appendix D presents the parallelization of the code, which can further reduce the time to solution.

4.1 A Baseline Dynamic Incentive Model

To address dynamic incentive problems with high-dimensional state spaces, we build on Fernandes and Phelan (2000) and consider a dynamic incentive model with persistent types.[23] These persistent types render the model's state space multidimensional, endogenous, and irregularly shaped, making it an ideal test case for the solution method proposed in this paper.

We consider infinite-horizon, discrete-time economies, where t represents the current period.[24] The model consists of a risk-averse agent with unobserved, persistent taste shocks and a risk-neutral planner who provides optimal, incentive-compatible insurance against taste shocks. The agent reports his type to the principal, who then transfers consumption to the agent, depending on the report. Since the principal can only see the reports, her optimal strategy must be fully history-dependent.

More formally, we assume that the principal and the agent have the same discount factor $\beta \in (0,1)$. Furthermore, the agent receives an idiosyncratic taste shock $\theta_t \in \Theta \subset \mathbb{R}$ in period t, and $U : C \times \Theta \to \mathbb{R}$ is his per-period utility function, and $C \subset \mathbb{R}$. The set Θ of taste shocks is discrete and finite with cardinality $|\Theta| = m$. To ensure the compactness, we also assume that the set of feasible consumption choices is a closed and bounded interval $C = [\underline{c}, \overline{c}]$ for some $\underline{c}, \overline{c} \in \mathbb{R}_0^+$.

The history of shocks is denoted by: $\theta^t = (\theta_1, \ldots, \theta_t) \in \prod_{i=1}^{t} \Theta = \Theta^t$. $\pi_t(\theta^t)$ denotes the probability of realization of the history θ^t. The agent learns his type at the beginning of period t and therefore has the information θ^t to condition his decisions on. The idiosyncratic shock θ_t evolves following a first-order Markov process:

$$\pi_t\left(\theta_t | \theta^{t-1}\right) = \pi\left(\theta_t | \theta_{t-1}\right), \forall \theta^{t-1} \in \Theta^{t-1}, \theta_t \in \Theta. \tag{13}$$

We denote $\pi_t(\theta^t \mid \theta^s)$ for $t > s$ as the probability that we observe θ^t given history θ^s. We set $\pi_t(\theta^t \mid \theta^s) = 0$ if θ^s is not identical to the first s components of θ^t. Without loss of generality, we assume that Θ is sorted in ascending order, and enumerate it as $\Theta = \{\theta_{(1)}, \ldots, \theta_{(m)}\}$ with $\theta_{(i)} < \theta_{(i+1)}$ for all i.

The principal chooses consumption allocations $c_t : \Theta^t \to C$. Below, we use the short-

[23] Compared to the original model, our primary modification in the numerical experiments presented below (cf. Section 4.5) is to increase the number of persistent taste shocks from two to up to ten.

[24] We follow the concise notation by Golosov et al. (2016) in describing the general setting of Fernandes and Phelan (2000).

hand notation c to denote $\left\{c_t\left(\theta^t\right)\right\}_{t\geq 1,\theta^t\in\Theta^t}$. The agent's period zero utility can now be written as

$$U_0(c) \equiv \mathbb{E}_0\left[\sum_{t=1}^{\infty}\beta^{t-1}U\left(c_t,\theta_t\right)\right] = \sum_{t=1}^{\infty}\sum_{\theta^t\in\Theta^t}\beta^{t-1}\pi_t\left(\theta^t\right)U\left(c_t\left(\theta^t\right),\theta_t\right), \tag{14}$$

where $\mathbb{E}_0$ is the ex-ante, "period-0" expectation before the initial shock θ_1 was realized.

Note that the principal does not observe the taste shock θ_t. Thus, she relies on reports from the agent. The agent's reporting strategy in time period t is given by $\tilde{\sigma}_t : \Theta^t \to \Theta$, and the principal chooses an allocation rule dependent on these reports $\tilde{c}_t : \Theta^t \to C$. Following the literature, we define $\tilde{\sigma} = \left\{\tilde{\sigma}_t\left(\theta^t\right)\right\}_{t\geq 1,\theta^t\in\Theta^t}$ and $\tilde{c} = \left\{\tilde{c}_t\left(\hat{\theta}^t\right)\right\}_{t\geq 1,\hat{\theta}^t\in\Theta^t}$. Taken together, they induce a measure over the consumption path, which is denoted as $\tilde{c}\circ\tilde{\sigma}$. We call $\tilde{\sigma}$ *incentive-compatible* if

$$\mathbb{E}^{\tilde{c}\circ\tilde{\sigma}}\left[\sum_{t=1}^{\infty}\beta^{t-1}U\left(c_t,\theta_t\right)\right] - \mathbb{E}^{\tilde{c}\circ\tilde{\sigma}'}\left[\sum_{t=1}^{\infty}\beta^{t-1}U\left(c_t,\theta_t\right)\right] \geq 0, \text{ for all strategies } \tilde{\sigma}', \tag{15}$$

where $\mathbb{E}^{\tilde{c}\circ\tilde{\sigma}}$ is the expectation with respect to the measure $\tilde{c}\circ\tilde{\sigma}$. In particular, if a reporting strategy is incentive-compatible, there is no deviation that improves the agent's payoff.

A critical insight necessary for analyzing the contracting environments we are interested in is the *revelation principle* (see, e.g., Golosov et al. (2016, Th. 1), and references therein). It states that for any incentive-compatible allocation, there is an incentive-compatible allocation with the same consumption path where the agent only reports truthfully. Thus, without loss of generality, we can simplify the comparison in Eq. (15) by ensuring that truth-telling is the optimal strategy. Accordingly, we replace Eq. (15) with:

$$\sum_{t=1}^{\infty}\sum_{\theta^t\in\Theta^t}\beta^{t-1}\pi_t\left(\theta^t\right)\left[U\left(c_t\left(\theta^t\right),\theta_t\right) - U\left(c_t\left(\sigma'^t\left(\theta^t\right)\right),\theta_t\right)\right] \geq 0 \quad \forall\sigma'. \tag{16}$$

This constraint ensures that truth-telling is incentive-compatible and, therefore, preferred over lying. The agent has a reservation utility that yields a utility amount $\underline{U}$. As a direct consequence, the principal has to offer more than the reservation utility, that is,

$$U_0(c) \geq \underline{U}. \tag{17}$$

We assume that the principal's preferences are given by the per-period utility function $v : C\times\Theta\to\mathbb{R}$. Thus, she solves the following optimization problem:

$$\begin{aligned} V(\underline{U}) \equiv \sup_{c} & \sum_{t=1}^{\infty}\sum_{\theta^t\in\Theta^t}\beta^{t-1}\pi_t\left(\theta^t\right)v\left(c_t\left(\theta^t\right),\theta_t\right) \\ & \text{subject to Eqs. (16), (17).} \end{aligned} \tag{18}$$

However, even after all these simplifications, the dynamic adverse selection problem under consideration remains intractable in its current form because it is fully history-dependent and, therefore, lacks a recursive formulation in its original state. We follow the literature and introduce promised utility vectors $(\hat{v}(\theta_{(1)}), \ldots, \hat{v}(\theta_{(m)})) \in \mathbb{R}^m$ as auxiliary state variables. At each point in time, these vectors represent the utility levels that each type of agent receives through direct transfers and promises for future payments. This approach yields a recursive formulation (see, e.g., Golosov et al., 2016, Section 2.5).

In the next step, we introduce the auxiliary variable $w(\hat{\theta}|\theta)$ to be the utility promise if $\hat{\theta}$ is reported, but θ actually happened. Written in a recursive form, we then obtain a new constraint called the *promise-keeping* constraint. It ensures that the utility promises $\hat{v}\left(\theta_{(1)}\right), \ldots, \hat{v}\left(\theta_{(m)}\right)$ that were made in the previous period are fulfilled by a combination of payments in the current period and future promises, that is,

$$\hat{v}\left(\theta_{(j)}\right) = \sum_{\theta \in \Theta} \pi\left(\theta|\theta_{(j)}\right) \left[U(c(\theta), \theta) + \beta w(\theta|\theta)\right], \quad \forall j \in \{1, \ldots, m\}. \tag{19}$$

The incentive compatibility constraint turns into

$$U(c(\theta), \theta) + \beta w(\theta|\theta) \geq U(c(\hat{\theta}), \theta) + \beta w(\hat{\theta}|\theta), \quad \forall \theta, \hat{\theta} \in \Theta, \tag{20}$$

which ensures that *one-shot* deviations from the truth-telling path are excluded. This condition ensures that the agent reports truthfully throughout.

Next, we need to establish that there is a state space $\mathcal{V} \subset \mathcal{B} \times \Theta$, where $\mathcal{B} \subset \mathbb{R}^m$, such that for any $\hat{v} = (\hat{v}\left(\theta_{(1)}\right), \ldots, \hat{v}\left(\theta_{(m)}\right), \vartheta)^T \in \mathcal{V}$, where $\vartheta \in \Theta$, there are $w(\cdot|\cdot)$[25] that satisfy the promise-keeping (cf. Eq. (19)) and incentive compatibility constraints (cf. Eq. (20)). Notice that since Θ is a discrete set with m elements, $\mathcal{V}$ is a collection of m sets, each of which are subsets of $\mathbb{R}^m$. Following Abreu et al. (1986, 1990), we define a set-valued mapping as

$$\begin{aligned}
\mathcal{A}\tilde{\mathcal{V}} = \Big\{ & (\hat{v}\left(\theta_{(1)}\right), \ldots, \hat{v}\left(\theta_{(m)}\right), \vartheta) \in \mathcal{B} \times \Theta \mid \\
& \hat{v}\left(\theta_{(j)}\right) = \sum_{\theta \in \Theta} \pi\left(\theta|\theta_{(j)}\right) \left[U(c(\theta), \theta) + \beta w(\theta|\theta)\right], \quad j = 1, \ldots, m \\
& U(c(\theta), \theta) + \beta w(\theta|\theta) \geq U(c(\hat{\theta}), \theta) + \beta w(\hat{\theta}|\theta), \quad \forall \theta, \hat{\theta} \in \Theta \\
& c(\theta) \in C, (w(\theta|\cdot), \theta) \in \tilde{\mathcal{V}}, \quad \forall \theta \in \Theta \Big\}.
\end{aligned} \tag{21}$$

From Abreu et al. (1986, 1990) and Golosov et al. (2016, Prop. 8), it is known that there exists a set-valued fixpoint $\mathcal{V}$, that is, $\mathcal{V} = \mathcal{A}\mathcal{V}$. This set constitutes the domain of the value function in our recursive dynamic programming problem and may exhibit irregular, non-hypercubic geometry as well as high dimensionality.

[25] In what follows, we use $w(\cdot \mid \cdot)$ as shorthand notation for the matrix $\left(w(\hat{\theta} \mid \theta)\right)_{\hat{\theta} \in \Theta, \theta \in \Theta}$, and $w(\hat{\theta} \mid \cdot)$ or similar to denote vectors.

In sum, the recursive formulation for the value function $K : \mathcal{V} \to \mathbb{R}$ of the dynamic incentive problem now reads as follows:

$$
\begin{aligned}
K(\hat{v}(\cdot), \theta_-) = \max_{\{c(\theta), w(\theta|\cdot)\}_{\theta\in\Theta}} & \sum_{\theta\in\Theta} \pi(\theta|\theta_-)\left[v(c(\theta), \theta_-) + \beta K(w(\theta|\cdot), \theta)\right] \\
& \text{subject to} \\
& \hat{v}\left(\theta_{(j)}\right) = \sum_{\theta\in\Theta} \pi\left(\theta|\theta_{(j)}\right)\left[U(c(\theta), \theta) + \beta w(\theta|\theta)\right], \quad j = 1, \ldots, m \\
& U(c(\theta), \theta) + \beta w(\theta|\theta) \geq U(c(\hat{\theta}), \theta) + \beta w(\hat{\theta}|\theta), \quad \forall \theta, \hat{\theta} \in \Theta \\
& c(\theta) \in C, (w(\theta|\cdot), \theta) \in \mathcal{V}, \quad \forall \theta \in \Theta,
\end{aligned}
\tag{22}
$$

where θ_- denotes the current discrete shock, $c(\cdot)$ and $w(\cdot|\cdot)$ are the controls, and $\hat{v}(\cdot)$ is the continuous state. Moreover, the state space is given by $\mathcal{V}$. Note that this problem no longer explicitly depends on time t; however, the unknown set $\mathcal{V}$ remains part of the constraints. More precisely, the issue is that the auxiliary problem stated in Eq. (22) contains promised utilities that are absent in the original formulation given in Eq. (18).

Lastly, note that the promise utilities are auxiliary variables for writing problem (18) in recursive form. Therefore, we describe in the following how to obtain the solution to the original problem from the solution to the auxiliary problem. The procedure we outline below will be used to obtain simulated paths for our proposed solution method. Given a seed type θ_0 for the Markov chain, we pick a state in $\mathcal{V}$ that maximizes the value function, that is,

$$
v_{max} \in \arg\max_{\hat{v}\in\mathcal{V}} K(\hat{v}, \theta_0). \tag{23}
$$

Then, given the sequence of shock realizations $\theta_0, \theta_1, \theta_2, \ldots$, the principal's optimal policy is recursively defined by setting initially

$$
\begin{aligned}
\hat{v}_0 &= v_{max}, \\
c_1 &= c(\theta_1), \\
\hat{v}_1 &= w(\theta_1|\cdot),
\end{aligned}
\tag{24}
$$

where $c(\theta_1)$ and $w(\theta_1|\cdot)$ are the optimal solutions to Eq. (22) given a state $(\hat{v}_0, \theta_0)$. Thus, at time $t = 1$, we solve problem (22) with the state $(\hat{v}_1, \theta_1)$, and again define

$$
\begin{aligned}
c_2 &= c(\theta_2), \\
\hat{v}_2 &= w(\theta_2|\cdot).
\end{aligned}
\tag{25}
$$

Proceeding iteratively generates a sequence of consumption transfers $c_1, c_2, \ldots$, representing the realization of the optimal solution to Eq. (18) for the given shock sequence.

4.2 A Numerically Tractable Reformulation of the Recursive Problem

Solving dynamic adverse selection models with discrete, persistent types in their recursive form presents significant challenges for the solution method: it requires solving a potentially high-dimensional dynamic programming problem on an a priori unknown, irregularly shaped set. If the problem is simple enough, it can be possible to find an analytical solution (Mailath et al., 2002). However, in general, one must resort to a numerical approach. Typically, the literature employs a two-step procedure:

1. Determine the feasible set using the APS algorithm (Abreu et al., 1986, 1990).
2. Solve a dynamic program over the domain identified in step 1.

However, this two-step procedure has notable limitations. It is often confined to specific model classes (e.g., those with convex feasible sets) and suffers from scalability issues due to the curse of dimensionality. Although several variants have been proposed in the literature, none provide a fully general solution: most restrict attention to two- or three-dimensional problems and cannot easily scale beyond that range (cf. Section 2).

To address these challenges, we propose a generic novel penalty reformulation of the Bellman equation for dynamic adverse selection problems. This reformulation offers three key advantages:

1. It eliminates the need to pre-determine the feasible set via the APS algorithm (Abreu et al., 1986, 1990).
2. It offers general applicability to dynamic adverse selection problems without restrictions to specific model classes, such as those with convex feasible sets.
3. It simplifies the solution process to solving an ordinary dynamic programming problem via value function iteration.

Our proposed approach leverages existing theory and practices for solving standard dynamic programming problems. Specifically, GPs and BAL are employed to approximate value and policy functions, facilitating efficient value function iteration.[26] This combination enables us to tackle dynamic incentive models with persistent discrete types of unprecedented dimensionality (cf. Section 4.5).

In what follows, we will introduce the reformulation in three steps. First, Section 4.2.1 provides the basic intuition behind our method. Second, Section 4.2.2 presents all the mathematical details, including two proofs that the solution of the relaxed model is a solution to the original problem. Third, Section 4.2.3 demonstrates how the reformulation can be mapped into classical dynamic programming.

[26] In addition to GPs, other machine learning methods, such as deep neural networks (see, e.g., Goodfellow et al. (2016)), have recently garnered the interest of computational economists for solving and estimating dynamic models. Appendix E outlines why, for the model class presented here, we consider GPs to be the more suitable choice.

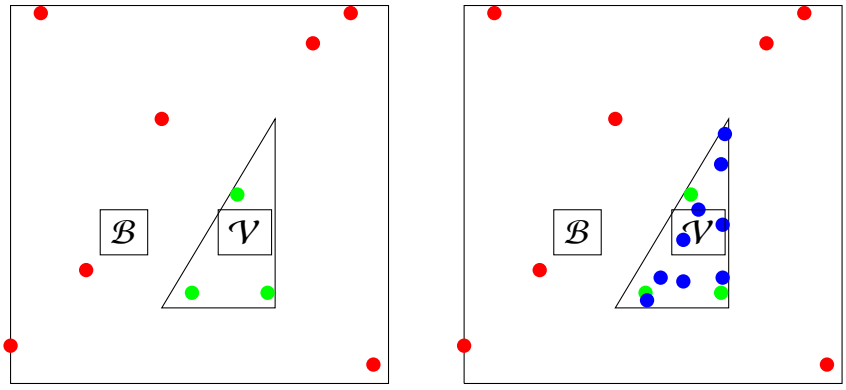

Figure 3: The left panel depicts a small set of sample points $\mathbf{X}$ that are uniformly drawn from the (hypercubic) set $\mathcal{B}$ that contains the *true* feasible set $\mathcal{V}$ (schematically depicted as a triangle). The red dots represent the points where the original Bellman operator (cf. Eq. (22)) would be deemed infeasible to solve, whereas the green points label locations in the state space where the solution to the original problem can be classified as feasible. The right panel shows the same situation as the left panel, but with the difference that the blue labeled points were systematically added with BAL to the original training set in the neighborhood of the points that were previously deemed feasible, thus focusing the computational resources where they are needed the most.

4.2.1 Intuition

The basic idea behind our reformulation is the following: While the feasible set $\mathcal{V}$ for the models under consideration is ex-ante unknown, we are usually able to give an estimate of a set $\mathcal{B}$ containing it, that is, $\mathcal{V} \subset \mathcal{B}$.[27] Thus, we start the value function iteration algorithm by uniformly drawing a moderate number of sample points $\mathbf{X}$ from the compact domain $\mathcal{B}$ at which we would like to evaluate the Bellman equation in order to obtain a training set $\mathcal{D}$ to approximate the value and policy functions with GPs. However, solving the original Bellman equation at these randomly drawn states may render some individual optimization problems infeasible, as illustrated in the left panel of Fig. 3. To this end, we propose to relax the original recursive model by introducing slack variables on the promise-keeping constraints. Furthermore, we also add a penalty to the objective function that is large and negative whenever the slacks are non-zero. This measure serves two purposes: First, this approach renders all optimization problems numerically feasible; specifically, it allows the evaluation of the relaxed Bellman operator on the entire domain $\mathcal{B}$, enabling the fitting of GPs on all training data, including infeasible points. Secondly, the

[27] For notational simplicity, we describe here a problem where the feasible set does not change depending on the taste shock. However, our methodology is not limited to this setting: if there are constraints that depend on the current shock θ–as in the model by Fernandes and Phelan (2000) that we consider in our numerical experiments below–so will the feasible set (cf. Section 4.5). Recall that in such situations, $\mathcal{V} = \{\mathcal{V}_{\theta_{(1)}}, \ldots, \mathcal{V}_{\theta_{(m)}}\}$, where each $\mathcal{V}_{\theta_{(i)}} \subset \mathbb{R}^m$ represents the subset of $\mathcal{V}$ corresponding to a particular value $\theta_{(i)} \in \Theta$.

relaxed problem provides a binary classifier of the training data, that is, $\mathcal{D} = \mathcal{D}_{\mathcal{V}} + \mathcal{D}_{\mathcal{B}\setminus\mathcal{V}}$. There are two situations where the relaxed Bellman equation can be deemed infeasible: The first is when the penalty function in the objective is non-zero. The second case is indirect: An infeasible state, that is, a state outside of $\mathcal{V}$, can have a zero penalty. The reason for this is that the promise-keeping and incentive constraints might be feasible, but only by providing infeasible future promises, that is, promises where the penalty is non-zero. However, these future promises will have a lower utility due to the penalty term. Thus, $\mathcal{D}_{\mathcal{V}}$ provides a first, rough approximation of the feasible set $\mathcal{V}$.[28] Next, to improve both the approximate value and policy functions on the *true* feasible set, we apply BAL to $\mathbf{X}_{\mathcal{V}}$ (the training points determined by $\mathcal{D}_{\mathcal{V}}$), systematically increasing the number of sample points in the region of interest[29] until global error estimates (cf. Eq. (36) in Section 4.3 below) become sufficiently small. Thus, a high-quality approximation of value and policy functions on an approximate feasible set is reached (cf. the right panel of Fig. 3).[30]

4.2.2 Mathematical Details of the Reformulation

After presenting the basic intuition behind our proposed reformulation, we now outline its mathematical details within the framework of the baseline model introduced in Section 4.1. We also provide two results regarding the penalty relaxation. Lemma 1 establishes the theoretical foundation for our relaxation approach, while Lemma 2 demonstrates its convergence in numerical applications. Specifically, Lemma 1 shows that, under certain conditions—which can be verified during the execution of the algorithm—the solution to the relaxed problem coincides with that of the original problem (cf. Eq. (18)). Furthermore, in Lemma 2, we prove that in the finite-horizon case, a stronger result holds: for a sufficiently large penalty, the optimal solution of the relaxed problem converges arbitrarily close to the optimal solution of the original problem. Hence, from a numerical perspective, the relaxed problem effectively solves the original problem when the penalty is sufficiently large.

In the first step, we need to determine a compact set $\mathcal{B}\times\Theta$ that contains $\mathcal{V}$ (cf. Fig. 3).[31] Recall from Section 4.1 that we assume that the consumption transfers c are bounded from above by $\bar{c}$, and that the agent's utility is bounded from below by $U(\underline{c},\theta)$. With these bounds, we can determine the following set from which we can draw sample points

[28] In practical applications, the cardinality of the initial draw of $\mathbf{X}$ needs to be sufficiently large such that $\mathcal{D}_{\mathcal{V}}$ is non-empty.

[29] In our applications, we perform BAL by generating candidate points along several simulation paths of 1,000 steps each that is started from the maximizers of the respective types. For more details, see Section 4.5.

[30] In practice, after every few steps of the value function iteration (cf. Section 4.4), we add new points to the training set via BAL while preserving the points from the previous iteration steps. However, the training targets, of course, have to be recomputed. For more practical details, see Sections 4.4 and 4.5.

[31] Note that our numerical approach can, in principle, also deal with unbounded sets. For such a situation, we merely require a good guess for the set to start from. However, in this situation, we lose the convergence guarantees from the theory of dynamic programming over compact sets (Stokey et al., 1989).

$\mathbf{X} = \{\mathbf{X}_{\theta_{(1)}}, \ldots, \mathbf{X}_{\theta_{(\mathbf{m})}}\}$, that is,

$$\mathcal{V} \subset \mathcal{B} \times \Theta = \left(\prod_{\tilde{\theta} \in \Theta} [U(\underline{c}, \tilde{\theta})/(1-\beta), U(\bar{c}, \tilde{\theta})/(1-\beta)] \right) \times \Theta, \tag{26}$$

Next, we need to replace the original recursive formulation with a penalized one that renders the Bellman equation feasible on the entire domain $\mathcal{B} \times \Theta$, and not solely on $\mathcal{V}$ (Luenberger and Ye, 2008, Chapter 13). To do so, we need to replace the set $\mathcal{V}$ with $\mathcal{B} \times \Theta$ in Eq. (22). Furthermore, we need to add additional variables and a penalty function to Eq. (22) to relax the promise-keeping that otherwise would be infeasible outside of $\mathcal{V}$ (cf. Fig. 3). Thus, we rewrite the Bellman equation as follows:

$$\begin{aligned}
\bar{K}(\hat{v}(\cdot), \theta_-) = \max_{c(\cdot), w(\cdot|\cdot), \xi_j} & \sum_{\theta \in \Theta} \pi(\theta|\theta_-) \left[v(c(\theta), \theta_-) + \beta \bar{K}(w(\theta|\cdot), \theta) \right] - M \sum_{j=1}^{m} \xi_j^2 \\
& \text{subject to} \\
& \hat{v}\left(\theta_{(j)}\right) + \xi_j = \sum_{\theta \in \Theta} \pi\left(\theta|\theta_{(j)}\right) [U(c(\theta), \theta) + \beta w(\theta|\theta)], \quad \forall j \in \{1, \ldots, m\} \\
& U(c(\theta), \theta) + \beta w(\theta|\theta) \geq U(c(\hat{\theta}), \theta) + \beta w(\hat{\theta}|\theta), \quad \forall \theta, \hat{\theta} \in \Theta \\
& c(\theta) \in C, w(\theta|\cdot) \in \mathcal{B} = \prod_{\tilde{\theta} \in \Theta} [U(\underline{c}, \tilde{\theta})/(1-\beta), U(\bar{c}, \tilde{\theta})/(1-\beta)], \quad \forall \theta \in \Theta,
\end{aligned} \tag{27}$$

where θ_- denotes again the current discrete shock, and $\bar{K}$ is the value function for the relaxed problem defined on $\mathcal{B} \times \Theta \subset \mathcal{V}$. It is also an upper bound on the true value function over $\mathcal{V}$. Furthermore, M is a non-negative constant,[32] and $\xi_j \in \mathbb{R}$ are slack variables. Notice that the reformulation of the model given by Eq. (27) renders the Bellman equation (numerically) feasible on the entire domain $\mathcal{B} \times \Theta$ by relaxing the promise-keeping constraints via the variables ξ_j, and by adding a penalty function $-M \sum_{j=1}^{m} \xi_j^2$ to the objective.

There are two ways to use the solution to the relaxed problem (27) to determine whether a point in the state space is infeasible (cf. the right panel of Fig. 3). The first way is by looking at the slack variables: if they are non-zero, the point can be deemed infeasible. The second way is to look at the level of the value function $\bar{K}$ of the relaxed problem. The original problem (22) has a natural minimum that reads as follows:

$$K_{min} = \sum_{\theta \in \Theta} \pi(\theta|\theta_-) \, v(\underline{c}, \theta_-)/(1-\beta). \tag{28}$$

[32] In practice, the choice of M depends on numerical experimentation. If, for a particular value of M, it is found that $\xi_j = 0$ along all simulated paths starting at the maximizer of the value function (cf. Eq. (24)), then we can conclude that M is sufficiently large, as implied by Lemma 1.

Thus, if the objective function of the relaxed model is lower than the said value at a particular location in the state space, a training point can be classified as infeasible once the value function iteration has converged.[33]

We now prove that the relaxed problem can be used to find the solution to the original problem (18). The following lemma demonstrates that, in general, the value function overestimates Eq. (22) and is exact at the maximizer under certain conditions.

Lemma 1. *The function $\bar{K}$ (given in Eq. (27)), is an overestimate of K (given in Eq. (22)) on the set $\mathcal{V}$. Furthermore, for any $\theta_0 \in \Theta$, let $\hat{v}_{max}(\theta_0) \in \arg\max_{\hat{v}(\cdot)\in\mathcal{B}} \bar{K}(\hat{v}(\cdot), \theta_0)$, and let $\hat{v}_{max}(\theta^t)$ be the utility promises given by the shock sequence $\theta^t = (\theta_0, \ldots, \theta_t)$ and the optimal policy of the relaxed problem (27). If for all infinite shock sequences $\theta^\infty = (\theta_0, \theta_1, \theta_2, \ldots)$ and for all $s \in \mathbb{N}$, all penalty variables are zero, that is, $\xi_i((\hat{v}_{max}(\theta_s), \theta_s) = 0$ for all $i = 1, \ldots, m$, then $K(\hat{v}_{max}(\theta_0), \theta_0) = \bar{K}(\hat{v}_{max}(\theta_0), \theta_0)$. In particular, the solution to the relaxed problem is a solution to the true problem. Specifically, if the optimal policy to the relaxed problem results in zero penalties for all shock realizations, then this relaxed policy provides an optimal solution to the original problem.*

Proof. Let us consider the optimal policy to the non-penalized problem (22), and augment it by $\xi_j(\hat{v}) = 0$ for all j and $\hat{v} \in \mathcal{V}$ then this is a feasible solution to the relaxed problem (27). Thus $K(\hat{v}) \leq \bar{K}(\hat{v})$ on all of $\mathcal{V}$. To show equality, that is, $K(\hat{v}_{\max}(\theta_0), \theta_0) = \bar{K}(\hat{v}_{\max}(\theta_0), \theta_0)$ for all $\theta_0 \in \Theta$, note that, by assumption, $(\xi_s)_j = 0$ for all $s \in \mathbb{N}$, $j = 1, \ldots, m$, and for all possible sequences. Furthermore, since $(\xi_s)_j = 0$, we know that the optimal policy for the relaxed problem (27) is feasible for the original problem (22), and therefore $\bar{K}(\hat{v}_{\max}(\theta_0), \theta_0) \leq K(\hat{v}_{\max}(\theta_0), \theta_0)$. Thus with $\bar{K} \geq K$ we get equality. □

Lemma 1 shows that the optimal value of the relaxed problem is equal to the original problem if the penalty is not invoked on any simulation path. But what about the optimal policy in the relaxed model? In this context, we can show a slightly weaker result, one that is valid for the finite time horizon version of the problem. In what follows, we will prove convergence of the value and policy functions on the entire feasible set $\mathcal{V}$ of the reformulated problem for a sufficiently large penalty parameter M and a finite time horizon.

Lemma 2. *Consider the finite time horizon version of the dynamic incentive problem (27), that is, $T < \infty$. Then, for all feasible initial promises $\hat{v}(\cdot)$ and every $\varepsilon > 0$, there exists a threshold $\tilde{M}$ such that for all $M > \tilde{M}$, the optimal solution of the penalty reformulation lies within a Euclidean distance of less than ε from the true solution.*

Proof. See Appendix F. □

[33] Note that while we do not have to approximate the feasible set within our reformulated model explicitly, we can still obtain it implicitly. When solving the model recursively, we approximate the value function and policies, including the slack variables, with GPs. Thus, looking either at the approximate slack variables or the value function itself will equip us with a smooth classifier that, on the entire set $\mathcal{B} \times \Theta$, will tell us approximately whether a location in the state space is feasible.

We require the result of Lemma 2 for stationary points as well, as these will ultimately be computed using standard numerical optimization. Fortunately, an analogous result holds for the stationary points, as demonstrated in Ben-Tal and Nemirovski (2023, Th.9.2.2). Thus, under non-degeneracy, stationary points in the penalty reformulation will correspond to stationary points in the KKT conditions of the original problem. For more details, see Ben-Tal and Nemirovski (2023).

In solving a dynamic programming problem numerically via value function iteration, we effectively address a finite-horizon version of the problem while assuming convergence of the value function in the limit. This ensures that our proposed algorithm—combining the relaxed dynamic optimization problem with value function iteration (cf. Sections 4.2.3 and 4.4)—as established through Lemma 2, converges to the true solution in a manner consistent with any numerical approach applied to solving dynamic programming problems.

After reformulating the recursive dynamic adverse selection problem into a numerically more tractable form, we next discuss how it can be solved using standard tools from the dynamic programming literature, such as value function iteration.

4.2.3 Dynamic Programming

The recursive problem stated in Eq. (27) is an ordinary dynamic programming problem defined over a compact set. Consequently, the classical theory (Stokey et al., 1989, Chapter 9) applies, ensuring the contraction property and, therefore, convergence. As a result, any general-purpose dynamic programming method for multi-dimensional problems (see, e.g., Cai and Judd, 2014, Maliar and Maliar, 2014, Brumm et al., 2022 and references therein) can be employed to solve this model recursively, for example, via value function iteration (for a textbook treatment, see, e.g., Bertsekas, 2000, Judd, 1998, Rust, 1996 or Ljungqvist and Sargent, 2000). The latter is motivated by the theoretical justification that iteratively applying the Bellman operator provides a sequence of value functions that converges to a unique fixed point. Value function iteration, in turn, numerically implements the iterative application of the Bellman operator to successive approximations of the value function. The corresponding dynamic programming recursion thus starts from any bounded and continuous guess for the value function $\bar{K}$ and the solution is approached in the limit as $\tau \to \infty$ by iterations on

$$\bar{K}^{\tau+1} = T(\bar{K}^{\tau}), \tag{29}$$

where the Bellman operator T, for instance, is given by the right-hand side of Eq. (27). In practice, we say that value function iteration has converged if numerical convergence in some norm, for example,

$$\|\bar{K}^{\tau} - \bar{K}^{\tau-1}\|_2 \leq \varepsilon_{\text{vfi}}, \quad \varepsilon_{\text{vfi}} > 0, \tag{30}$$

and at some iteration step τ is reached. The (approximate) equilibrium value function is denoted as $\bar{K}^*$ and the corresponding optimal policy functions, as $\chi^* = \arg\max \bar{K}^*$.

To link the dynamic adverse selection problem (27) to the notation commonly used in the dynamic programming literature more explicitly, we define the following two quantities:

$$r_v(c(\theta_{(1)}), \dots, c(\theta_{(m)}), \xi_1, \dots, \xi_m, \theta_-) = \sum_{j=1}^{m} \pi(\theta|\theta_-)\, v(c(\theta_{(j)}), \theta_-) - M\xi_j^2, \tag{31}$$

where r_v is shorthand for the reward in the current period. In iteration step τ, we need to solve the following optimization problem at many points in the state space:

$$\begin{aligned}
\bar{K}^{\tau}(\hat{v}(\cdot), \theta_-) = \max_{c(\cdot), w(\theta|\cdot), \xi_j} \; & r_v(\cdot, \theta_-) + \beta \mathbb{E}_\theta \left[\bar{K}^{\tau-1}(w(\theta|\cdot), \theta) \mid \theta_- \right] \\
& \text{subject to} \\
& \hat{v}\left(\theta_{(k)}\right) + \xi_k = \sum_{j=1}^{m} \pi\left(\theta_{(j)}|\theta_{(k)}\right) \left[U(c(\theta_{(k)}), \theta_{(k)}) + \beta w(\theta_{(j)}|\theta_{(j)}) \right], \\
& U(c(\theta_{(i)}), \theta_{(i)}) + \beta w(\theta_{(i)}|\theta_{(i)}) \geq U(c(\theta_{(k)}), \theta_{(i)}) + \beta w(\theta_{(k)}|\theta_{(i)}), \\
& c(\theta_{(k)}) \in C, w(\theta_{(k)}|\cdot) \in \mathcal{B} = \prod_{j=1}^{m} [U(\underline{c}, \theta_{(j)})/(1-\beta), U(\bar{c}, \theta_{(j)})/(1-\beta)], \\
& \forall k \in \{1, \dots m\}, \quad \forall i \in \{1, \dots m\},
\end{aligned} \tag{32}$$

where $\bar{K}^{\tau-1}(w(\theta|\cdot), \theta)$ is the interpolation value of the value function of the previous iteration $\tau - 1$ at a given location in the state space, and where Π refers to the Cartesian product. To further connect the dynamic adverse selection model to the dynamic programming literature, notice that the vector of current states can be written as

$$x = (\hat{v}(\theta_{(1)}), \dots, \hat{v}(\theta_{(m)}), \theta_-) \in \mathcal{B} \times \Theta, \tag{33}$$

whereas the vector of future states can be abbreviated as

$$x^+ = (w(\theta^+|\theta_{(1)}), \dots, w(\theta^+|\theta_{(m)}), \theta^+) \in \mathcal{B} \times \Theta, \tag{34}$$

assuming θ^+ is the next period's discrete state. Lastly, the set of control variables, whose cardinality we will abbreviate in the following with $\kappa = |\chi|$, reads as follows:

$$\chi = \left\{ c(\theta_{(i)}), \xi_i, w(\theta_{(i)}|\theta_{(j)}) \mid i, j = 1, \dots m \right\}. \tag{35}$$

In Section 4.4, we outline the implementation of the value function iteration procedure described here, leveraging GPs combined with BAL to efficiently approximate and interpolate the value function (and policy functions, when required) at each iteration step.

4.3 Validation, Verification, and Uncertainty Quantification

Since our models typically lack closed-form analytical solutions and must be determined numerically, it is crucial to ensure that the obtained results are credible, correct, and reliable. To achieve this, we adhere to the best practices of *validation, verification, and uncertainty quantification* (VVUQ), a sub-field of computational science that provides structured methodologies and metrics to assess the accuracy and reliability of numerical solutions (Oberkampf and Roy, 2010). Within the VVUQ framework, we employ four criteria that jointly establish the numerical quality of our solution. In particular, the first criterion we present is the termination criterion for the value function iteration at some iteration step τ:

1. **Termination Criterion Based on BAL:** We employ the BAL acquisition function (cf. Eq. (10)) to determine when further improvements in the GP approximations become negligible. Specifically, we track the acquisition score of the "best-ranked" candidate point across types and terminate once $\alpha(\cdot;\cdot) < \epsilon_{BAL}$, with no further significant decrease observed over several value function iteration steps. This criterion indicates that no substantial new information is being incorporated. Using the "elbow" logic commonly adopted in machine learning (see, e.g., Murphy, 2022), exceeding this threshold corresponds to diminishing marginal returns; that is, the marginal gains from adding new information become negligible. Since BAL candidate points are generated via simulation when solving dynamic incentive models (cf. Section 4.4), satisfying this criterion implies that the approximated policy functions have stabilized and are effectively time-invariant. This ensures the model is numerically solved with high accuracy.
2. **Global Convergence Check:** Beyond ensuring that there are no further gains from BAL, we also verify the contraction property of the value function iteration over the entire domain $\mathcal{B} \times \Theta$ (cf. Eq. (30)). Once this global error measure is sufficiently small, it confirms that the approximate value function is stable and accurate across the entire state space.
3. **Local Accuracy Check Along the Optimal Policy:** Global convergence alone does not ensure that the solution is sufficiently accurate in the state space regions most relevant to the optimal policy. To address this, we track the value function error *along the simulated path of the optimal policy*, following approaches in Haan et al. (2011) and Azinovic et al. (2022). Specifically, starting from the initial shock type and using the corresponding value function maximizer, we simulate the optimal policy to obtain a sequence of states $\hat{v}^{(1)}, \ldots, \hat{v}^{(s)}$ with associated shocks $\theta^{(1)}, \ldots, \theta^{(s)}$. At these states, we compare consecutive GP approximations of the value function, $K^{\tau-1}$ and K^{τ}, at the final iteration step τ:

$$\epsilon_{sim} = \frac{1}{s}\sqrt{\sum_{l=1}^{s}\left(\frac{\bar{K}^{\tau}(\hat{v}^{(l)},\theta^{(l)}) - \bar{K}^{\tau-1}(\hat{v}^{(l)},\theta^{(l)})}{1+|\bar{K}^{\tau}(\hat{v}^{(l)},\theta^{(l)})|}\right)^2}. \tag{36}$$

By normalizing the error, we ensure comparability across the entire problem domain. This criterion verifies that the solution is accurate where it matters most, that is, along the trajectories the model actually uses.

4. **Feasibility Check:** At this stage, we evaluate an additional convergence measure to confirm that the computed solutions belong to the feasible set. Specifically, Lemma 1 provides a metric verifying that the policy $\xi_i = 0$ is preserved throughout the simulated paths. We conduct multiple simulations, starting from the maximum of the value function over the promise utility space (with varying random seeds), and confirm that feasibility is maintained. This ensures the reported results are derived exclusively from feasible policies, enhancing the credibility of the final solution.

Practical Implementation: In our numerical experiments (cf. Sections 4.5 and 5), we set ϵ_{BAL} to a value determined through model-dependent hyperparameter tuning. For example, when solving our benchmark model (cf. Section 4.5.1), we set $\epsilon_{BAL} \approx 1 \cdot 10^{-2}$ and terminate the value function iteration once this threshold is reached (Criterion 1). At the same time, we continuously verify that the global (Criterion 2) and local (Criterion 3) metrics decrease to ensure contraction and convergence, while confirming feasibility (Criterion 4).[34] By applying these four criteria in unison, we ensure that the final solution is numerically stable, reliable, and suitable for subsequent analysis.

4.4 A Solution Algorithm for Dynamic Incentive Problems

We now detail how to numerically solve the reformulated dynamic adverse selection models with a standard dynamic programming approach by combining value function iteration with GPs (cf. Section 3.1), BAL (cf. Section 3.2), and parallel computation (cf. Appendix D).[35]

As discussed in Section 4.3, four criteria ensure the numerical quality of the computed solutions: (1) termination based on the BAL acquisition score, (2) global convergence, (3) local accuracy along the simulated optimal policy, and (4) feasibility.

The detailed algorithm we propose for computing optimal decision rules in dynamic incentive problems is presented in Alg. 1 in Appendix A and can be summarized as follows: We begin by drawing, for every discrete state present in the model (that is, every type θ), a uniform sample of J points $\mathbf{X}_\theta = \{\hat{v}_\theta^{(1)}, \ldots, \hat{v}_\theta^{(J)}\} \subset \mathcal{B}$, the hypercube of promise utilities (cf. Eq. (26)).[36] At each of these points, we provide an initial guess $\bar{K}_\theta^{init}(\hat{v}_\theta^{(i)})$ for the value func-

[34] Once Criterion 1 is met and Criterion 4 is also satisfied, the relative global errors associated with Criteria 2–3 are typically negligible (of order $O(10^{-5})$ or smaller). In such cases, even if one continues iterating the value function to achieve even lower thresholds, it does not change the implications of the computed solutions, making further refinement unnecessary.

[35] The value function iteration algorithm proposed in this section is loosely related to that proposed by Keane and Wolpin (1994), who use Monte Carlo simulations and linear regression to solve and estimate discrete choice dynamic programming problems, a setting that substantially differs from the one we are targeting here.

[36] We use $\hat{v}_\theta^{(i)}$ as shorthand notation for $(\hat{v}^{(i)}, \theta)$, that is, the location of the training input i at which the

tion. Jointly, these form the initial training set $\mathcal{D}_\theta^{init} = \{(\hat{v}_\theta^{(1)}, \bar{K}_\theta^{(1),init}), ..., (\hat{v}_\theta^{(J)}, \bar{K}_\theta^{(J),init})\}$, over which we fit a GP per type θ. With this setup, we commence the value function iteration.

In every iteration step τ, we fit a GP $\mathcal{G}_\theta^\tau$ to the current training set $\mathcal{D}_\theta^\tau$. This step is highly parallelizable and can be substantially accelerated by contemporary hardware (cf. Appendix D). We then apply BAL to concentrate computational effort on the most relevant regions of the state space. By generating a set of candidate points $\mathcal{U}_\theta$ through simulations initiated from each type's maximizer, we score these candidates using, for example, Eq. (10). We add the $\mathcal{P}$ points with the highest scores (often one or two) to the training set $\mathbf{X}_\theta$ and solve the model-implied Bellman equations (cf. Eqs. (27)) at these new points to obtain updated targets $\{\bar{K}_\theta^{(1),\tau}, \ldots, \bar{K}_\theta^{(J+\mathcal{P}),\tau}\}$, over which we fit new GPs.[37]

This iterative process continues until the four VVUQ criteria (cf. Section 4.3) are simultaneously satisfied. Otherwise, the iteration continues. At the point where all four criteria are met, we have a numerical solution of acceptable accuracy for the approximate value functions $\bar{K}^* = \{\bar{K}_{\theta_{(1)}}^\tau, \ldots, \bar{K}_{\theta_{(m)}}^\tau\}$ and the corresponding κ policy functions per state (cf. Eq. 35), that is, $\chi^* = \{\chi_{\theta_{(1)},1}^\tau, \ldots, \chi_{\theta_{(1)},\kappa}^\tau, \ldots, \chi_{\theta_{(m)},1}^\tau, \ldots, \chi_{\theta_{(m)},\kappa}^\tau\}$, each given by the predictive mean of a GP. For further implementation details, we refer the reader to the code available at the following URL: https://github.com/GaussianProcessesForDynamicEcon/DynamicIncentiveProblems.

Finally, note that one of the defining features of GPR is its grid-free nature, enabling the modeler to closely steer the composition of the training set $\mathcal{D}$ and construct interpolators on irregular geometries. This offers two significant advantages for solving dynamic incentive models numerically. First, if an individual optimization problem at a specific point x_i fails to converge, the corresponding input and nonsensical target can simply be excluded from $\mathcal{D}$. This eliminates the need for tuning the optimizer at that location. In contrast, grid-based methods, such as Smolyak (see, e.g., Krueger and Kubler, 2004, Judd et al., 2014) or adaptive sparse grids (see, e.g., Brumm and Scheidegger, 2017, Brumm et al., 2022) fail when optimization problems at required grid points cannot be solved, leading to a breakdown of the interpolator's construction. Second, solving only on the domain of relevance allows value function iteration on complex, high-dimensional geometries without inefficiencies. Computational resources are concentrated where needed, avoiding unnecessary effort in regions where the GP interpolant is already accurate. This significantly reduces the time-to-solution, particularly in high-dimensional settings where the feasible set may represent a negligible fraction of the computational domain required by standard methods.

Bellman equation is evaluated. Furthermore, the training inputs can be written as $\mathbf{X} = \{\mathbf{X}_{\theta_{(1)}}, \ldots, \mathbf{X}_{\theta_{(\mathbf{m})}}\}$, and the training set as $\mathcal{D} = \{\mathcal{D}_{\theta_{(1)}}, \ldots, \mathcal{D}_{\theta_{(m)}}\}$.

[37] In our numerical experiments (see Sections 4.5 and 5), each optimization problem at a training point is solved with `Ipopt` (see, e.g., Waechter and Biegler, 2006) or Scipy's constrained optimization toolbox, yielding identical results up to numerical precision and providing a robustness check for our solutions.

4.5 Performance and Accuracy

To evaluate the performance and scalability of our computational framework, we revisit the dynamic adverse selection model introduced in Section 4.1. In Section 4.5.1, we use the baseline calibration by Fernandes and Phelan (2000) as a fundamental test case, where approximate numerical solutions are (at least graphically) available for a two-type version of the model. This low-dimensional setting enables clear visualization of the solutions, providing a direct benchmark for assessing the accuracy of our proposed dynamic programming method. After demonstrating that our algorithm can accurately reproduce known results in the two-type setting with high precision, we test its scalability to higher dimensions. In Section 4.5.2, we significantly increase the dimensionality of the model by extending the benchmark model to ten persistent types. We first present the parameterization of this ten-dimensional model and then evaluate the algorithm's performance in this considerably more demanding environment, confirming that the solutions remain accurate. This showcases the method's capability to reliably handle significantly more complex problems while consistently delivering accurate results.

4.5.1 Results for the Dynamic Incentive Problem with Two Types

Model. The environment we adopt from Fernandes and Phelan (2000) features a risk-neutral principal who *minimizes* her cost, and a risk-averse agent. The agent's utility over consumption, c, is defined as

$$U(c) = 2\sqrt{c + r} - 2\sqrt{r}, \tag{37}$$

where c is constrained to a compact interval, and r is a small regularization term introduced for numerical stability.[38] The model distinguishes between two types: a "low" type (state 1, also denoted as L) and a "high" type (state 2, also denoted as H), with respective endowments h_L and h_H. The agent learns his private type in each period and reports it to the principal, who then transfers consumption based on the agent's reported type. As the problem depends on the full history of reports, we employ the recursive reformulation introduced earlier (cf. Eq. (27)) to address the numerical difficulties previously discussed. Following Fernandes and Phelan (2000), we assume that the agent cannot claim to be of a higher type than his true type. This assumption is implemented by removing the corresponding incentive compatibility constraints:

$$U(c(\theta_{(i)}) + \theta_{(i)}) + \beta w(\theta_{(i)}|\theta_{(i)}) \geq U(c(\theta_{(k)}) + \theta_{(i)} - \theta_{(k)}) + \beta w(\theta_{(k)}|\theta_{(i)}), \tag{38}$$

where $\theta_{(i)}$ is the true type and $\theta_{(k)}$ is the reported type. Whenever $\theta_{(i)} < \theta_{(k)}$, as in the two-state case where $\theta_{(i)} = \theta_L$ and $\theta_{(k)} = \theta_H$, the corresponding constraint is omitted. This

[38] The utility function, augmented with a regularization term, is a slight modification of the formulation in Fernandes and Phelan (2000). This term was introduced solely to enhance numerical stability, deviating from the original utility function $\sqrt{c}$. Numerical instability arises because the slope of the utility tends to infinity as consumption c approaches zero. The regularization term mitigates this issue.

Parameter	Value
β	0.9
h_H	0.35
h_L	0.1
$[\underline{c}, \overline{c}]$	$[0, 1]$
$\mathcal{B}$	$[0, 10.0]^2$
r	10^{-8}
M	100
ϵ_{BAL}	$1 \cdot 10^{-2}$

Table 1: Parameterization of the privately observed endowment model.

eliminates the issue that $c(\theta_{(k)}) + \theta_{(i)} - \theta_{(k)}$ may become negative, rendering the square root utility function U used here undefined.

Since this model consists of two persistent types, the resulting state space of promised utilities is 2-dimensional. The Markov process governing the endowments is chosen such that the agent has a 90 percent chance of receiving the endowment he received in the previous period. This yields the following transition probabilities across the different types:

$$\Pi = \begin{bmatrix} \pi(L|L) & \pi(H|L) \\ \pi(L|H) & \pi(H|H) \end{bmatrix} = \begin{bmatrix} 0.9 & 0.1 \\ 0.1 & 0.9 \end{bmatrix}. \tag{39}$$

The remaining model parameterization is reported in Tab. 1.

To solve this benchmark model, we initiate the dynamic programming algorithm (cf. Alg. 1) by sampling 64 points for each type θ from the hyper-cubic domain $\mathcal{B} = [0, 10.0]^2$ using a Halton sequence. We use the following initial guess for the value function at each sampled point:

$$\bar{K}_\theta^{\text{init}} = \frac{\pi(L|\theta)(h_L - U^{-1}(\hat{v}_\theta(L)(1-\beta))) + \pi(H|\theta)(h_H - U^{-1}(\hat{v}_\theta(H)(1-\beta)))}{1-\beta}, \tag{40}$$

where $\theta \in \{L, H\}$. GPs are initialized using piecewise polynomial kernels ($q = 0$) fitted on the initial training set $\mathcal{D}$ to launch the value function iteration. Training data is standardized at each step of the value function iteration.[39] During the solution process, the training set is systematically enhanced using BAL. Candidate points are generated

[39]To enhance the numerical stability of the training process, we standardize the training inputs, as is common practice in the GP literature. Additional techniques that can improve training performance and may be applied in other contexts include the following: standardizing targets to match an initial GP variance of 1; setting the initial noise level of the GP to a high value, even if the data appear to have low noise, as this improves the optimization surface for other hyperparameters; and performing multiple random restarts when maximizing the GP likelihood to avoid local optima during hyperparameter optimization.

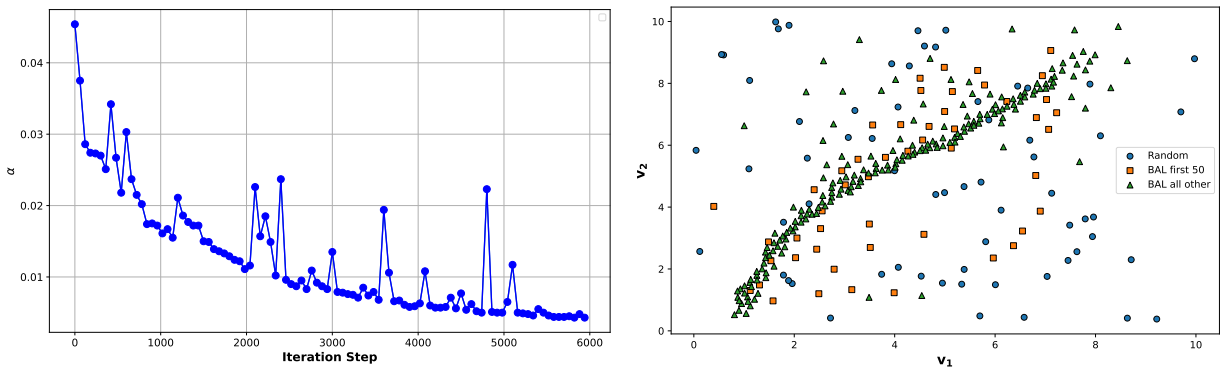

Figure 4: Left panel: Criterion 1 ("BAL") as a function of value function iteration steps. Right panel: The initial set $\mathbf{X}_{h_L}$, comprising 64 points (blue) dots, is sampled from $\mathcal{B}$ using a Halton sequence and systematically enhanced via BAL. The first 50 added points are represented as (orange) squares, while subsequent points are shown as (green) triangles. The points are adaptively added in regions where the model simulation predominantly evolves, illustrating the effectiveness of the BAL process.

along a simulation path of 1,000 steps, initialized at the maximizer.[40] Due to the numerical expense of generating candidate points, only one point per type is added to the training set every 30 steps of the value function iteration (cf. Section 4.4).[41]

Performance. Now, we provide evidence that our solution method delivers highly accurate solutions by systematically applying the four convergence criteria introduced in Section 4.3.

To begin, we focus on Criterion 1. The left panel of Fig. 4 shows how the acquisition score evolves as new points are added to the training set throughout the value function iteration procedure. In the right panel, we observe how the training set (of the low type) is iteratively enriched with points where they improve the quality of the GP approximation

[40] In this section, we use the BAL acquisition function from Wagberg et al. (2017), which is slightly more sophisticated than the simpler function discussed in Eq. (10), as it provided the best results. However, the baseline acquisition function in Eq. (10), with hyperparameters $\sigma_m = 1$ and $\sigma_v = 4$, yielded results of nearly the same quality.

[41] At the end of the value function iteration process (i.e., at convergence), each iteration step takes roughly 22 seconds on a standard desktop using 4 MPI processes (AMD Ryzen 9). In this final phase, a total of 522 optimization problems (for both types combined) must be solved per iteration, making the reported runtime an upper, conservative bound. Earlier in the iteration process, only about 25% of these points are used, which—given the roughly quadratic scaling in the number of optimization problems (Gardner et al., 2018) —reduces the runtime by an order of magnitude per iteration step. Moreover, parallelization can easily accelerate the procedure by about two orders of magnitude per iteration step if proportionally more CPUs are employed. Of course, these numbers are provided as illustrative examples only, as the actual total runtime depends on various additional factors, including software implementation details, hardware, the initial guess for the value function, and the parametrization of the model, such as the discount factor β. Nevertheless, a model like that presented here can typically be solved in less than half an hour using hardware readily available to most users.

Error type	L_2	L_∞
Criterion 2 (global error)	$9.6 \cdot 10^{-6}$	$3.6 \cdot 10^{-5}$
Criterion 3 (error along a simulated path)	$4.7 \cdot 10^{-7}$	$5.6 \cdot 10^{-5}$

Table 2: Average and maximum relative errors at convergence. The first error type, "global error,"was computed by evaluating Eq. (36) at 1,000 uniformly drawn sample points on $\mathcal{B}$, whereas "error along a simulated path"was computed by generating the same number of observations along a simulated path and that was launched at the point in the state space given by $\arg\max_{\hat{v}(\theta),\theta} \bar{K}(\hat{v}(\theta), \theta)$.

most. Together, these figures highlight the importance of BAL: by concentrating the GP approximation on the equilibrium path, it effectively mitigates the curse of dimensionality and allocates computational effort where it matters most. Indeed, as the iteration progresses, the sample points cluster in a particular region of the state space, and the marginal benefits of adding more points diminish. Once each $\mathcal{D}_\theta$ contains about 250 points per type, the improvements from BAL become negligible, indicating that Criterion 1 is satisfied. At this stage, the value and policy functions are stable and would not change substantively even if we continued to refine the solution further.

Next, we examine Criteria 2 and 3, which quantify errors over the entire computational domain $\mathcal{B}$ and along a simulated path, respectively. Using 1,000 sample points, we compute the average and maximum errors and report them in Tab. 2. These results confirm the excellent quality of our solution. Although we have performed value function iteration until further improvements became virtually negligible, typical applications would not require such extreme accuracy (e.g., values as low as $O(10^{-7})$), as the economic implications of the solutions remain largely unchanged beyond a precision threshold that is likely 1–2 orders of magnitude higher. With Criterion 1 satisfied and Criteria 2–3 showing no further improvements, we proceed to Criterion 4 to ensure that the computed solutions are feasible. To this end, we simulate the model by conducting 64 independent simulations. Each simulation starts at the state-space point that maximizes the value function and involves 1,000 random shocks drawn from the distribution Π (cf. Eq. (39)). All resulting 64,000 points remain within the feasible set, satisfying Criterion 4. In Fig. 5, we illustrate the simulation trajectory for the low type, which starts in the lower-left region and quickly moves toward the top-right boundary of the feasible set.[42] Notably, large portions of the state space are never visited, underscoring that BAL places dense sampling precisely where it is needed most.

Results. In the simulation depicted in Fig. 5, the optimal policy begins in the lower-left corner of the feasible set and gradually transitions toward the upper-right region. The gaps

[42]This figure illustrates the feasible set computed using the auxiliary problem where the utility is set to zero. In this special case, the feasible set consists of all points where the value function equals zero. To create the plot, we uniformly sampled approximately 250 points per type from $\mathcal{B}$, classified them as feasible or infeasible, and then plotted a contour line separating $\mathcal{V}$ (the feasible region) from the infeasible part.

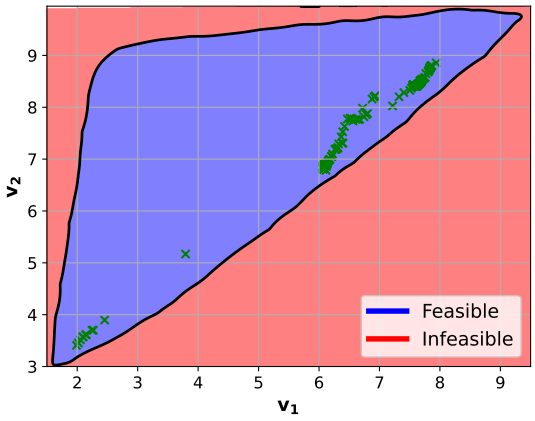

Figure 5: This figure illustrates the trajectory of a simulation (green circles) representing the optimal policy as a function of $\hat{v}(L)$ and $\hat{v}(H)$ (denoted by $\mathbf{v_1}$ and $\mathbf{v_2}$), with the approximate feasible set for the low type overlaid for context.

in the trajectory represent transitions between the two agent types. The principal's strategy initially involves offering low promises to both agent types, deferring their fulfillment far into the future. This strategy emerges because the low discount factor in this model (cf. Tab. 1) allows for larger future promises without significantly diminishing the current payoff.

Comparing our results against the earlier findings of Fernandes and Phelan (2000), we observe that the feasible set we obtain visually aligns with their reported findings.[43] Taken together, the figures and performance metrics presented in this section confirm that our dynamic programming approach effectively solves dynamic incentive problems with high accuracy. Building on this foundation, we proceed in Section 4.5.2 to address models with more than two types, exploring higher-dimensional settings.

4.5.2 Results for the Dynamic Incentive Problem with Ten Types

Model. After thoroughly demonstrating the effectiveness of our solution method on a classical benchmark model, we now briefly illustrate its scalability to higher-dimensional state spaces by tackling dynamic incentive problems with more than two persistent types. Specifically, we extend the stylized environment described in Section 4.5.1 to a case with ten persistent types. This results in a ten-dimensional state space of promised utilities. Unless otherwise noted, the setup follows the two-type benchmark, with only the modifications required for the ten-type specification highlighted in the text below.

While our primary objective is to demonstrate the scalability of our approach beyond

[43] Note that Fernandes and Phelan (2000) do not provide quantitative convergence measures, policy functions, or computational run-times, so our comparison is necessarily limited to visual inspection of value functions and approximate feasible sets.

Parameter	Value
β	0.9
h_{θ_1}	0.1
$h_{\theta_{10}}$	0.35
h_{θ_i}	$h_{\theta_1} + (i-1)(h_{\theta_{10}} - h_{\theta_1})/9$ for $i = 2, \ldots, 9$
$[\underline{c}, \overline{c}]$	$[0, 1]$
$\mathcal{B}$	$[0, 10.0]^{10}$
r	10^{-8}
M	50
ϵ_{BAL}	0.3

Table 3: Parameterization of the privately observed endowment model with ten types.

the commonly studied two-type setting (see Golosov et al., 2016 for a review), extending models similar to the one used here to incorporate many more types can also be relevant for real-world applications, such as unemployment insurance systems in various countries with multiple tiers of benefits.

We index the types by $i = 1, \ldots, 10$, where $i = 1$ corresponds to the type with the lowest endowment and $i = 10$ to the type with the highest. In line with the literature, we assume that the agent can only underreport their type. For example, if the agent is type 5, they can only report being types 1 to 5. The Markov process governing the endowments is defined by the following transition matrix:

$$\Pi = \begin{bmatrix} 0.9 & 0.1 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\ 0.1 & 0.8 & 0.1 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0.1 & 0.8 & 0.1 & 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0.1 & 0.8 & 0.1 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0.1 & 0.8 & 0.1 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0.1 & 0.8 & 0.1 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0.1 & 0.8 & 0.1 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 & 0.1 & 0.8 & 0.1 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0.1 & 0.8 & 0.1 \\ 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0.1 & 0.9 \end{bmatrix}. \tag{41}$$

The choice of transition matrix straightforwardly extends the original shock structure. It is designed to facilitate comparisons between the original two-dimensional benchmark version of the model (cf. Section 4.5.1) and the larger ten-dimensional version considered here. The remaining parameters of the model are reported in Tab. 3.

To solve the ten-dimensional model, we follow Algorithm 1 with two minor adjustments: instead of initializing value function iteration by uniformly drawing points from the hypercube $\mathcal{B}$, we adopt a more targeted strategy. Because the feasible set is extremely small relative to the volume of $\mathcal{B}$, uniform sampling would be highly inefficient (only a

tiny fraction of the sampled points would lie in the feasible region), thus requiring a prohibitively large initial sample. To this end, we propose an alternative sampling strategy that places the initial samples closer to the true feasible set, which can then be augmented using BAL. The details are outlined in Appendix G. Furthermore, as a direct consequence, error Criterion 2 is evaluated using sample points generated as described in Appendix G, rather than through uniform sampling from $\mathcal{B}$. We initialize the value function iteration with 32 samples per type. Finally, we apply the following initial guess for the value function at each sampled point:

$$K_{init}(\hat{v}(\cdot), \theta_{(i)}) = \frac{1}{1-\beta} \sum_{\theta} \pi(\theta|\theta_{(i)}) u^{-1}((1-\beta)\hat{v}(\theta)). \tag{42}$$

where $i = 1, \ldots, 10$. We initialize the GPs by fitting piecewise polynomial kernels ($q = 0$) to the initial training set $\mathcal{D}$, and we standardize the training data at each iteration for numerical stability. As the value function iteration proceeds, we again systematically enrich the training set via BAL. Specifically, we generate candidate points along a $1,000$-step simulation path, starting from the maximizer. In the ten-dimensional case considered here, we employ again the BAL acquisition function by Wagberg et al. (2017), as it performed the best. As in previous sections, due to computational constraints, only one candidate point per type is added every 30 steps of the value function iteration.[44]

Performance. We now provide evidence that our solution method delivers highly accurate results in high-dimensional settings by systematically applying the four convergence criteria introduced in Section 4.3.

Focusing initially on Criterion 1, we terminate the value function iteration once the acquisition score drops below $\epsilon_{BAL} = 0.3$. By that stage, the training set contains $2,610$ points, or roughly 260 points per type. Because the improvements from BAL become negligible beyond this threshold, Criterion 1 is met. Moreover, the value and policy functions are already stable and would not change substantially with further iterations.

Next, we examine Criteria 2 and 3. Using 1,000 sample points, either from the algorithm in Appendix G or from simulation, we compute the average and maximum errors, as reported in Tab. 4. These results confirm the excellent quality of our solution, with errors as low as $O(10^{-6})$, even in this ten-dimensional case. The errors are only slightly higher

[44] A typical value function iteration step requires approximately 54 seconds on a single compute node with 64 MPI processes (Intel Ice Lake). During this time, about $2,600$ optimization problems (for all ten types combined) must be solved. Compared to the two-dimensional baseline model, this higher runtime reflects several factors: five times as many functions must be approximated, the number of training points is larger, and each optimization problem is more complex due to additional constraints and more states. Nonetheless, the overall compute time has only increased by a factor of about 40, which is significantly lower than the explosive growth one might expect from a naive approach suffering from the curse of dimensionality (e.g., eight orders of magnitude compared to the two-type benchmark when using a Cartesian grid-based method). As discussed earlier, actual runtimes depend on various factors and can be further reduced by approximately two orders of magnitude through parallel computing. Consequently, a model of this complexity can typically be solved within half a day to a day on standard hardware.

Error type	L_2	L_∞
Criterion 2 (global error)	$1.8 \cdot 10^{-5}$	$5.0 \cdot 10^{-5}$
Criterion 3 (error along a simulated path)	$5.6 \cdot 10^{-6}$	$1.0 \cdot 10^{-5}$

Table 4: Average and maximum relative errors at convergence for the ten-dimensional model. The "global error" was computed by evaluating Eq. (36) at 1,000 sample points generated as described in Appendix G, while the "error along a simulated path" was computed using the same number of observations along a simulated path, starting from the state-space point given by $\arg\max_{\hat{v}(\theta),\theta} \bar{K}(\hat{v}(\theta), \theta)$.

than in the two-dimensional case, mainly because the value function iteration was stopped once the policies became time-invariant according to Criterion 1.

With Criterion 1 satisfied and Criteria 2–3 showing no further improvements, we proceed to Criterion 4 to verify the feasibility of the computed solutions. To this end, we run 64 independent simulations, each starting at the state-space point that maximizes the value function and including 1,000 random shocks drawn from the distribution Π (cf. Eq. (3)). All 64,000 resulting points remain within the feasible set, confirming that Criterion 4 is satisfied.

Results. Tab. 5 reports the maximum value function for each of the 10 types. Notably, the payoff (i.e., the maximum value function per type), for type 1 is not the lowest; instead, it decreases until type 4 and then increases thereafter. Consequently, if the agent's initial type is 4, the principal obtains the overall lowest payoff. Note that the number of incentive constraints has increased significantly, from just one in the two-state model to 45 in the 10-type model,[45] naturally reducing the overall payoff. Moreover, since the model disallows overreporting, the number of constraints decreases for lower agent types. In particular, there are no incentive constraints for a type 1 agent, whereas higher types are associated with more constraints, raising the cost of incentive compatibility. This effect is most apparent for type 4, where the payoff to the principal is lower than for types with smaller endowments. For comparison, in the two-state benchmark model from the previous section, the payoffs for the low and high types are 0.12 and 0.13, respectively, indicating that payoffs increase with the agent's type. Therefore, the higher payout in the second state exceeds the cost of incentivizing the type-2 agent to report the true state.

θ	1	2	3	4	5	6	7	8	9	10
$\max \bar{K}$	-0.18	-0.27	-0.31	-0.33	-0.31	-0.25	-0.17	-0.05	0.03	0.09

Table 5: The table shows the maximum value of the value function per type, that is, $\max_{\hat{v}(\theta),\theta} \bar{K}(\hat{v}(\theta), \theta)$.

[45] In the model under consideration, if the number of types is m, the number of constraints is given by $\frac{1}{2}(m^2 - m)$.

5 Adverse Selection with Limited Overreporting

To demonstrate the broad applicability of our method, we now solve a multi-dimensional dynamic incentive model, with persistent types, that differs in important ways from the model discussed above. Specifically, we consider an adverse selection model with up to four discrete types and (limited) overreporting.

A common and often simplifying assumption in the dynamic incentive literature is that the agent cannot report a higher shock than the one received, an assumption we also followed in Section 4.5. Ruling out overreporting can be interpreted, for instance, as prohibiting secret borrowing outside the contract or preventing covert trade and production (e.g., Williams, 2011, Bloedel et al., 2024, and references therein). However, overreporting can be crucial to specific adverse selection problems, as it encompasses a broader range of misreporting strategies, capturing more realistic incentives and behaviors, such as individuals overstating their assets to obtain loans or firms exaggerating their performance to attract investors. It is also particularly relevant for understanding long-run contract outcomes, which often polarize into "immiseration" or "bliss" in many different models.

We contribute to this debate by studying such models in discrete time with multiple persistent types; problems that, to the best of our knowledge, have not yet been solved in the literature. In particular, we aim to investigate whether the overreporting assumption affects the long-run behavior of the contract. Concretely, we posit a model that builds on the framework from Section 4.5 and introduce limited overreporting, where the agent does not require outside help to overreport their state. Specifically, we implement this by activating the incentive constraint only when the contract allows the agent to overreport their type without relying on external assistance.

Beyond its economic relevance, the overreporting problem significantly increases the computational burden on the solution algorithm compared to the benchmark model discussed above, making it an ideal test case for our method. As we will show below, since overreporting is feasible only in certain regions of the feasible part of the state space, the resulting model becomes a complementarity problem, introducing non-differentiability into the constraint set.

This section is organized into two main steps. First, Section 5.1 introduces a model of limited overreporting with two persistent types, building on the benchmark model discussed in Section 4.5.1. We apply our solution method to this model, demonstrate its performance, and present the key findings. Next, Section 5.2 extends the model to incorporate four types and examines how the results differ from those of the two-type model.

5.1 A Model with Limited Overreporting: Two Types

Model. The stylized adverse selection model with overreporting that we propose builds on Fernandes and Phelan (2000). Accordingly, we focus on describing the deviations from this benchmark model necessary to incorporate overreporting. For clarity, we outline

the model of overreporting without penalty relaxation, which can be added in analogy to Section 4.2.

As before, we consider a discrete-time setup with a risk-neutral principal who minimizes her costs and a risk-averse agent. The agent's utility over consumption, c, is given by expression (37). For now, the model distinguishes between two types: a "low" type (state 1, denoted as L) and a "high" type (state 2, denoted as H), with respective endowments h_L and h_H, where $\theta_L < \theta_H$. The agent learns his private type in each period and reports it to the principal, who then transfers consumption based on the reported type.

Recall from Section 4.5.1 that in the standard adverse selection model, the incentive constraints associated with overreporting are omitted. To consider a model where the agent must be prevented from over-reporting his type, the following constraint has to be imposed:

$$U(c(\theta_L)) + \beta w(\theta_L|\theta_L) \geq U(\theta_L - \theta_H + c(\theta_H)) + \beta w(\theta_H|\theta_L). \tag{43}$$

However, expression (43) introduces a potential problem: if $\theta_L - \theta_H + c(\theta_H)$ is negative, it is outside U's domain for standard utility functions (such as the square-root utility used here). Hence, simply reintroducing the constraint is not a viable solution. The issue of $\theta_L - \theta_H + c(\theta_H)$ being negative is also the reason why interpreting overreporting as hidden borrowing makes sense because by blanketly allowing overreporting the agent can be seen to remedy this shortfall by using outside help.

We interpret this issue as follows. The agent receives an endowment of θ_L and contemplates what to report. Fully aware of the contract, the agent knows that misreporting θ_H would result in an immediate net transfer of $c(\theta_H) - \theta_H$ from the principal. Consequently, the agent would only consider misreporting if the following condition holds:

$$\theta_L + c(\theta_H) - \theta_H \geq 0. \tag{44}$$

Under this interpretation, there is no need for secret borrowing to facilitate overreporting. This behavior can be represented by reformulating expression (43) using the following complementarity condition:

$$\begin{aligned} &\max\{0, \varepsilon + \theta_L - \theta_H + c(\theta_H)\} \\ \cdot &\Big(U(c(\theta_L)) + \beta w(\theta_L \mid \theta_L) - \big(U(\theta_L - \theta_H + c(\theta_H)) + \beta w(\theta_H \mid \theta_L)\big)\Big) \;\geq\; 0, \end{aligned} \tag{45}$$

where (45) is active only if

$$\max\{0, \varepsilon + \theta_L - \theta_H + c(\theta_H)\} \;>\; 0, \tag{46}$$

indicating that overreporting is feasible. Note that $\varepsilon > 0$ is a parameter introduced to handle cases where the net transfer is zero. Put differently, expression (46) "switches on" the constraint in (43) whenever the contract allows the agent to benefit from misreporting, and "switches it off" when the contract terms themselves prevent overreporting.

Parameter	Value
β	0.99
ℓ_L	6
ℓ_H	12
h_L	0.65
h_H	1.35
$[\underline{c}, \bar{c}]$	$[0, 2]$
$\mathcal{B}$	$[0, 280]^2$
r	10^{-8}
M	100
ε	10^{-4}

Table 6: Parameterization of the adverse selection model with limited overreporting.

For the two-type problem considered in this section, we obtain the following recursive formulation for the dynamic incentive problem with persistent types and overreporting:

$$
\begin{aligned}
K\left(\hat{v}(\cdot), \theta_{-}\right) = \max_{c(\cdot), w(\cdot|\cdot)} & \sum_{\theta \in \Theta} \pi\left(\theta|\theta_{-}\right)\left[\theta - c(\theta) + \beta K(w(\theta|\cdot), \theta)\right] \\
& \text{subject to} \\
& \hat{v}\left(\theta_{(j)}\right) = \sum_{\theta \in \Theta} \pi\left(\theta|\theta_{(j)}\right)\left[U(c(\theta)) + \beta w(\theta|\theta)\right], \quad \forall j \in \{L, H\}, \\
& U(c(\theta_H)) + \beta w(\theta_H|\theta_H) \geq U(\theta_H - \theta_L + c(\theta_L)) + \beta w(\theta_L|\theta_H), \\
& \max\{0, \varepsilon + \theta_L - \theta_H + c(\theta_H)\} \\
& \quad \cdot \Big(U(c(\theta_L)) + \beta w(\theta_L|\theta_L) - \big(U(\theta_L - \theta_H + c(\theta_H)) + \beta w(\theta_H|\theta_L)\big)\Big) \geq 0, \\
& c(\theta) \in [\underline{c}, \bar{c}], \quad w(\theta|\cdot) \in \mathcal{V}.
\end{aligned}
\tag{47}
$$

Note that Eq. (47) includes an additional inequality constraint that is only non-trivial when the contract allows overreporting, that is, $\theta_L - \theta_H + c(\theta_H) \geq 0$. Consequently, the resulting optimization problem inherently has a value less than or equal to that of the baseline model discussed in the previous section.

To enhance the interpretability of the Markov chain parameters, we rewrite the transition matrix as:

$$
\Pi = \begin{bmatrix} \pi(L|L) & \pi(H|L) \\ \pi(L|H) & \pi(H|H) \end{bmatrix} = \begin{bmatrix} 1 - \frac{1}{\ell_L} & \frac{1}{\ell_L} \\ \frac{1}{\ell_H} & 1 - \frac{1}{\ell_H} \end{bmatrix}, \tag{48}
$$

where ℓ_L and ℓ_H represent the average durations the chain spends in states L and H, with realizations h_L and h_H, respectively. The remaining parameter values of the model are provided in Tab. 6.

Model Version	Error type	L_2	L_∞
Benchmark	Criterion 2 (global error)	$6.0 \cdot 10^{-7}$	$3.5 \cdot 10^{-6}$
Benchmark	Criterion 3 (error along a simulated path)	$6.8 \cdot 10^{-7}$	$1.1 \cdot 10^{-6}$
Overreporting	Criterion 2 (global error)	$1.6 \cdot 10^{-6}$	$2.6 \cdot 10^{-5}$
Overreporting	Criterion 3 (error along a simulated path)	$1.5 \cdot 10^{-6}$	$7.9 \cdot 10^{-6}$

Table 7: Average and maximum relative errors at convergence for the two-dimensional benchmark model (denoted as "Benchmark") and the model with overreporting (denoted as "Overreporting"), both parameterized according to Tab. 6. The "global error" was computed by evaluating Eq. (36) at 1,000 sample points uniformly drawn from $\mathcal{B}$, whereas the "error along a simulated path" was computed by generating the same number of observations along a simulated path launched from the state-space point given by the maximize of the value function.

To solve this model, we initiate the dynamic programming algorithm (cf. Alg. 1) by sampling 64 points for each type θ from the hyper-cubic domain $\mathcal{B} = [0, 280]^2$ using a Halton sequence. We use the same initial guesses for the value functions as in the benchmark model, provided in Eq. (40). GPs are initialized using piecewise polynomial kernels ($q = 0$) fitted on the initial training set $\mathcal{D}$ to launch the value function iteration. Additionally, the training data is standardized at each step of the value function iteration and sequentially enhanced using BAL, as described in Section 4.5.1.[46]

Performance. After presenting the model, we now briefly demonstrate that our solution method delivers highly accurate results for this more intricate model setup. At convergence, the training set consists of approximately 360 points in total, corresponding to about 180 points per type. The errors for Criteria 2 and 3 were computed using 1,000 sample points and are reported in Tab. 7. The errors for the model with overreporting are of very good quality (as low as $\mathcal{O}(10^{-6})$), particularly given that the GPs need to approximate policy functions with kinks. These results are contrasted with solutions to a two-type benchmark model (cf. Section 4.5.1) with the same parameterization but without overreporting. Our findings indicate that the performance of our method barely deteriorates, even in the presence of strong nonlinearities. Finally, we consider Criterion 4 to ensure that the computed solutions are feasible. As before, we conduct 64 independent simulations of the model. Each simulation begins at the state-space point that maximizes the value function and includes 1,000 random shocks drawn from the distribution Π (cf. Eq. (48)). All resulting 64,000 points remain within the feasible set, satisfying Criterion 4. In summary, these results confirm that our method is capable of producing solutions of excellent quality, even in the presence of strong nonlinearities.

[46] For the model with two types and overreporting, a typical value function iteration step close to convergence takes approximately 26 seconds on a single compute node with 64 MPI processes (Intel Ice Lake).

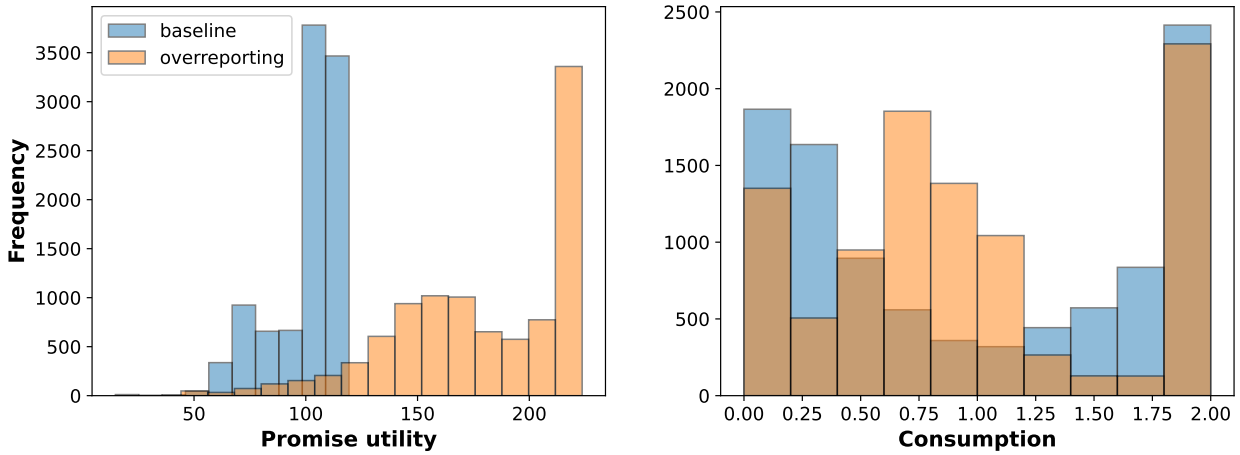

Figure 6: The left panel shows the promise utility for a truth-telling agent over a 10,000-period simulation (excluding the first 100 "burn-in" periods) for both the baseline and overreporting models. The right panel depicts the corresponding consumption transfer. Since $h_H = 1.35$ and $h_L = 0.65$, the overreporting constraint activates when $c \geq h_H - h_L = 0.7$.

Results. Having demonstrated that our method successfully solves dynamic incentive problems with overreporting, we now examine the economic implications of the two-type model. Specifically, we analyze how the long-run contract behavior varies under different assumptions imposed on the problem. To do so, we compute the model's ergodic distribution by simulating 10,000 steps, starting from the state-space point that maximizes the value function. To mitigate dependence on the initial condition, we discard the first 100 steps.

Fig. 6 presents the results, comparing the ergodic distributions of promise utilities and consumption in the baseline and overreporting models, yielding four key insights. First, allowing the agent to overreport produces a visibly different utility distribution: although both models exhibit a mass point on the right side, the maximum utility promise in the baseline model is roughly half that of the overreporting model. Neither model unambiguously displays "immiseration" or "bliss" when considering the utility promise alone, similar to the i.i.d. case (Golosov et al., 2016, Lemma 2).

Second, the baseline model features two mass points for consumption: one at the lower bound and one at the upper bound.

Third, the overreporting model exhibits three mass points, with fewer instances at the bounds, indicating that allowing overreporting drastically alters the distribution of long-run outcomes.

Finally, these findings demonstrate that limited overreporting significantly modifies the contract's long-run behavior, underscoring that leaving out any kind of overreporting is an impactful assumption.

5.2 A Model with Limited Overreporting: Four Types

Model. After demonstrating the effectiveness of our solution method on a two-dimensional adverse selection model with (limited) overreporting, we now illustrate its scalability to higher-dimensional state spaces by addressing dynamic incentive problems with four persistent types, resulting in a four-dimensional state space of promised utilities. The setup largely follows the two-type model in Section 5.1, with only the necessary modifications for the four-type specification highlighted below.

We start extending the two-type model from the previous section to higher dimensions by assuming that the shock takes the following form:

$$U(c(\hat{\theta}), \theta) = u(\theta - \hat{\theta} + c(\hat{\theta})). \tag{49}$$

Under this specification, the type of shock alters the relevant complementarity condition. For the given shock structure, the corresponding optimization problem is:

$$\begin{aligned}
K\left(\hat{v}(\cdot), \theta_{-}\right) = \max_{\{c(\theta), w(\theta|\cdot)\}_{\theta\in\Theta}} & \sum_{\theta\in\Theta} \pi\left(\theta|\theta_{-}\right)\left[v(c(\theta), \theta_{-}) + \beta K(w(\theta|\cdot), \theta)\right] \\
& \text{subject to} \\
& \hat{v}\left(\theta_{(j)}\right) = \sum_{\theta\in\Theta} \pi\left(\theta|\theta_{(j)}\right)\left[U(c(\theta), \theta) + \beta w(\theta|\theta)\right], \quad j = 1, \ldots, m \\
& \max\left\{0, \varepsilon + \theta - \hat{\theta} + c(\hat{\theta})\right\} \cdot \\
& \quad \left[U(c(\theta), \theta) + \beta w(\theta|\theta) - \left(U(c(\hat{\theta}), \theta) + \beta w(\hat{\theta}|\theta)\right)\right] \geq 0, \quad \forall \theta < \hat{\theta}, \\
& U(c(\theta), \theta) + \beta w(\theta|\theta) \geq U(c(\hat{\theta}), \theta) + \beta w(\hat{\theta}|\theta), \quad \forall \theta > \hat{\theta}, \\
& c(\theta) \in C, \ (w(\theta|\cdot), \theta) \in \mathcal{V}, \quad \forall \theta \in \Theta.
\end{aligned} \tag{50}$$

We then reformulate this problem as a complementarity problem by introducing multipliers $\mu_{\theta\hat{\theta}}$ whenever $\theta < \hat{\theta}$, resulting in:

$$\begin{aligned}
K\left(\hat{v}(\cdot), \theta_{-}\right) = \max_{c(\theta), w(\theta|\cdot), \mu_{\theta\hat{\theta}}} & \sum_{\theta\in\Theta} \pi\left(\theta|\theta_{-}\right)\left[v(c(\theta), \theta_{-}) + \beta K(w(\theta|\cdot), \theta)\right] \\
& \text{subject to} \\
& \hat{v}\left(\theta_{(j)}\right) = \sum_{\theta\in\Theta} \pi\left(\theta|\theta_{(j)}\right)\left[U(c(\theta), \theta) + \beta w(\theta|\theta)\right], \quad j = 1, \ldots, m \\
& \mu_{\theta\hat{\theta}} - \left(\varepsilon + \theta - \hat{\theta} + c(\hat{\theta})\right) \geq 0, \quad \forall \theta < \hat{\theta}, \\
& \mu_{\theta\hat{\theta}}\left[U(c(\theta), \theta) + \beta w(\theta|\theta) - \left(U(c(\hat{\theta}), \theta) + \beta w(\hat{\theta}|\theta)\right)\right] \geq 0, \quad \forall \theta < \hat{\theta}, \\
& U(c(\theta), \theta) + \beta w(\theta|\theta) \geq U(c(\hat{\theta}), \theta) + \beta w(\hat{\theta}|\theta), \quad \forall \theta > \hat{\theta}, \\
& c(\theta) \in C, \ (w(\theta|\cdot), \theta) \in \mathcal{V}, \quad \forall \theta \in \Theta, \\
& \mu_{\theta\hat{\theta}} \geq 0, \quad \forall \theta < \hat{\theta}.
\end{aligned} \tag{51}$$

Next, we extend the two-type model to four types by defining the shocks as:

$$h_i = h_L + i\,\frac{h_H - h_L}{3}, \quad i = 0, 1, 2, 3, \tag{52}$$

and specifying the transition matrix as:

$$\Pi = \begin{bmatrix} \alpha(1-\frac{1}{\ell_L}) & (1-\alpha)(1-\frac{1}{\ell_L}) & \frac{\alpha}{\ell_L} & \frac{1-\alpha}{\ell_L} \\ (1-\alpha)(1-\frac{1}{\ell_L}) & \alpha(1-\frac{1}{\ell_L}) & \frac{1-\alpha}{\ell_L} & \frac{\alpha}{\ell_L} \\ \alpha\frac{1}{\ell_H} & \frac{1-\alpha}{\ell_H} & \alpha\big(1-\frac{1}{\ell_H}\big) & (1-\alpha)\big(1-\frac{1}{\ell_H}\big) \\ \frac{(1-\alpha)}{\ell_H} & \frac{\alpha}{\ell_H} & (1-\alpha)\big(1-\frac{1}{\ell_H}\big) & \alpha\big(1-\frac{1}{\ell_H}\big) \end{bmatrix}, \tag{53}$$

with $\alpha = 0.75$. The chosen matrix retains a structure similar to the original, facilitating comparison with the 2D case.

To solve this model, we initiate the dynamic programming algorithm (cf. Alg. 1) by sampling 32 points per type θ, following the strategies outlined in Section 4.5.2 and Appendix G. We also use the same initial guesses for the value functions and retain all other GP and BAL specifications.[47]

Performance. At convergence, the training set consists of approximately 670 points in total, with around 160 points per type. The errors for Criteria 2 and 3 were computed using 1,000 sample points and are reported in Tab. 8. The errors for the model with overreporting remain of high quality, particularly given that the GPs approximate four-dimensional functions with kinks. These results are compared to a four-type benchmark model with the same parameterization but without overreporting. Criterion 4 is evaluated as in previous sections, confirming that all 64,000 simulated points remain within the feasible set. Thus, our findings reaffirm the capability of our method to achieve highly accurate solutions, even in the presence of strong nonlinearities and non-trivial dimensionality.

Results. We now investigate the economic implications of the four-type overreporting model, paying particular attention to how its long-run contract outcomes differ from those in simpler settings.

Fig. 7, which plots the distribution of promise utilities (left panel) and consumption transfers (right panel) over a 10,000-period simulation (discarding the first 100 periods), reveals several key observations. First, in the baseline model, the distribution of utility promises is noticeably flatter compared to the two-type setting (see Fig. 6), indicating that no single promise value dominates in higher dimensions.

Second, the overreporting model now yields a more concentrated distribution of utility promises, suggesting that the principal assumes greater risk while smoothing out the agent's utility promises over time.

[47] For the model with four types and overreporting, a typical value function iteration step near convergence takes approximately 56 seconds on a single compute node with 64 MPI processes (Intel Ice Lake).

Model Version	Error type	L_2	L_∞
Benchmark	Criterion 2 (global error)	$3.8 \cdot 10^{-7}$	$1.9 \cdot 10^{-6}$
Benchmark	Criterion 3 (error along a simulated path)	$3.4 \cdot 10^{-5}$	$1.9 \cdot 10^{-4}$
Overreporting	Criterion 2 (global error)	$8.3 \cdot 10^{-5}$	$1.9 \cdot 10^{-4}$
Overreporting	Criterion 3 (error along a simulated path)	$1.5 \cdot 10^{-6}$	$6.5 \cdot 10^{-6}$

Table 8: Average and maximum relative errors at convergence for the four-dimensional benchmark model ("Benchmark") and the model with overreporting ("Overreporting"). The "global error" was computed by evaluating Eq. (36) at 1,000 sample points generated using the strategy in Appendix G, while the "error along a simulated path" was computed from the same number of observations along a simulated path, starting from the state-space point that maximizes the value function.

Third, in terms of consumption transfers, the baseline model shifts away from extreme values relative to the two-type case. In contrast, the overreporting model clusters transfers around the mean consumption transfer, effectively eliminating high transfers.

Fourth, these differences underscore how both dimensionality and the ability to overreport can substantially alter the ergodic distribution, reinforcing the need for computational methods that handle higher-dimensional problems with complex constraints.

6 Conclusion

This article makes a series of contributions that enable the study of dynamic incentive models with heterogeneous agents and persistent shocks, that is, for high-dimensional models where the feasible set is not known ex-ante.

First, we introduce a penalty-based reformulation of dynamic incentive problems that is numerically easier to handle than the standard recursive formulation commonly used in the literature, as it allows us to bypass the explicit computation of the feasible set. Second, we provide formal proof that this reformulation converges to the same solution as the original model. Third, we propose a scalable and flexible value function iteration algorithm that combines Gaussian process regression with Bayesian active learning for solving a wide range of high-dimensional dynamic programming problems, including–but not limited to–those studied in this paper. Fourth, to demonstrate the capabilities of our framework, we apply it to the dynamic adverse selection model by Fernandes and Phelan (2000) as a verification test for our method, as the approximate numerical solutions are known for their two-dimensional baseline setting, and the results can be compared. Furthermore, since the models we study no longer have analytical solutions but have to be determined iteratively, the final ingredients for our framework are measures to assess the credibility and correctness of our computational results. To do so, we follow the best practices in a sub-field of computational sciences called *validate, verify, and uncertainty quantification*, and propose to use four particular error criteria jointly. Fifth, we introduce an adverse selection

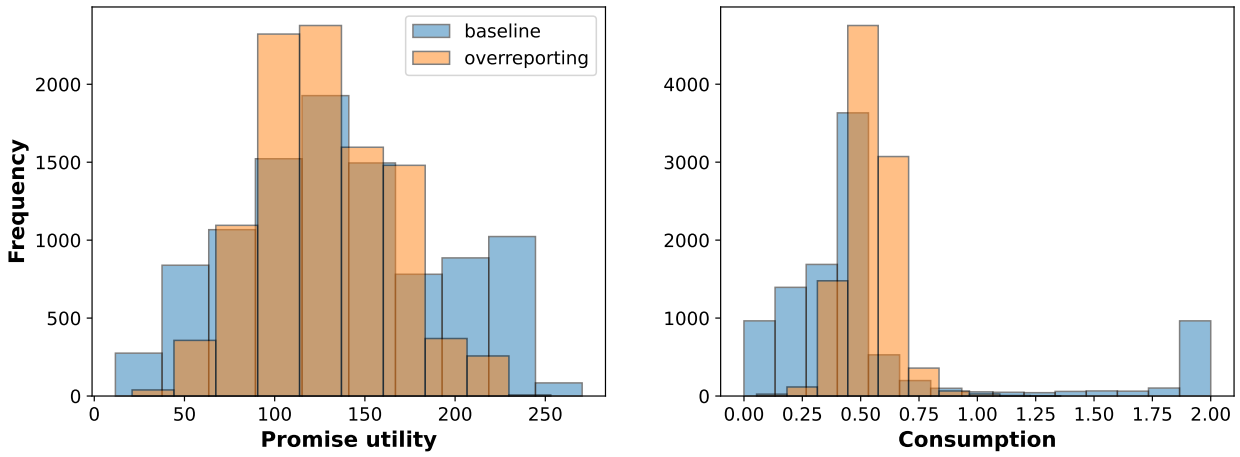

Figure 7: The left panel shows the promise utility for a truth-telling agent over a 10,000-period simulation (discarding the first "burn-in" 100 periods) for both the standard and the overreporting model. The right panel shows the corresponding consumption transfer. Note that we can no longer determine by visual inspection where the overreporting constraint becomes active. We find that at least one overreporting constraint was active in every iteration.

model with limited overreporting, enhancing the realism of the contract. This feature allows the agent to overreport his type when possible without requiring external assistance; an aspect previously overlooked in the literature. Overreporting introduces additional complementarity constraints, making the policy functions nonsmooth and necessitating a flexible function approximation framework like the one presented here. Our findings indicate that incorporating overreporting in a four-dimensional model significantly reduces the volatility faced by the agent, a feature absent in lower-dimensional settings or without overreporting. In addition, all our methodological developments are supplemented by a Python-based toolbox that can be found under the following URL: https://github.com/GaussianProcessesForDynamicEcon/DynamicIncentiveProblems.

In summary, this all suggests that our proposed framework will enable researchers to study dynamic incentive problems of a greater richness than was possible prior to this work, as they no longer need to drastically restrict their modeling choices from the outset.

Moreover, we emphasize that while the focus of the work presented in this paper lies in solving dynamic adverse selection problems with discrete, persistent shocks, the methods proposed here, being generic, have a far broader scope: They can prove useful in solving models where one or several of the following features occur: the state space is endogenous, of irregular geometry, or multi-dimensional. These situations can arise in problems such as:

- Dynamic adverse selection with a continuum of hidden states. If one applies the first-order approach to tackle such models, it becomes necessary to track the utility promise and the marginal utility promise as part of the state space. Furthermore, not

all promises are feasible.

- Dynamic moral hazard problems with multiple agents. Such models require keeping track of the utility promises to every agent. If the principal is constrained, for example, via a budget, then the state space becomes endogenous, as not all states will be affordable, that is, there are promises that cannot be paid with current cash on hand.
- Dynamic games. In these models, one needs to keep track of the history in order for the players to punish or reward the past behavior of their opponents. This history dependence also has a recursive formulation using promised utilities as an endogenous state space.
- Problems with debt and endogenous default. These models typically require that the researcher endogenously determines the borrowing limits as part of the model.

A Algorithm Details

Algorithm 1: Value Function Iteration with GPs, BAL, and VVUQ Criteria.

Data: For all θ, draw a (small) initial set of J points $\mathbf{X}_\theta = \{\hat{v}_\theta^{(1)}, \ldots, \hat{v}_\theta^{(J)}\} \subset \mathcal{B}$ such that it contains $\mathcal{V}_\theta$. Initial guess for the value functions K_θ^{init}. Set termination threshold ϵ_{BAL} for the BAL criterion (Criterion 1).

Result: For each θ: GPs over $\mathcal{B}$ for the value functions $\bar{K}_\theta^*$ and the κ policy functions per type.

Define:
- Criterion 1: BAL termination criterion $\alpha(\cdot;\cdot) < \epsilon_{BAL}$.
- Criterion 2: Global convergence check via Eq. (30).
- Criterion 3: Local accuracy check along simulated paths via Eq. (36).
- Criterion 4: Feasibility check via Lemma 1.

For each type θ: Generate initial training set $\mathcal{D}_\theta^{init} = \{(\hat{v}_\theta^{(1)}, \bar{K}_\theta^{(1),init}), ..., (\hat{v}_\theta^{(J)}, \bar{K}_\theta^{(J),init})\}$.
For each type θ: Fit a GP $\mathcal{G}_\theta$ over $\mathcal{D}_\theta^{init}$.
Set iteration step $\tau = 1$.
while *Not all four criteria are simultaneously satisfied* **do**
  for $\theta \in \Theta$ **do**
    Set $\mathcal{G}_\theta^{\tau-1} = \mathcal{G}_\theta$.
    Use predictive mean $\tilde{\mu}_\theta^{\tau-1}$ of $\mathcal{G}_\theta^{\tau-1}$ as an interpolator of $\bar{K}_\theta^{\tau-1}$.
    Use predictive variance $\tilde{\sigma}_\theta^{\tau-1}$ of $\mathcal{G}_\theta^{\tau-1}$ as a pointwise measure of uncertainty around $\bar{K}_\theta^{\tau-1}$.
    if $\tau \mod n = 0$ **then**
      Given $\tilde{\mu}_\theta^{\tau-1}$ and $\tilde{\sigma}_\theta^{\tau-1}$, perform BAL on the approximate feasible points contained in $\mathbf{X}_\theta$ to obtain a set $\mathcal{U}_\theta$ of s_θ candidate points.
      Add the $\mathcal{P}$ points with the largest BAL score (cf. Eq. (10)) to $\mathbf{X}_\theta$.
      Update $J = J + \mathcal{P}$.
    end
    foreach $\hat{v}_\theta^{(i)} \in \mathbf{X}_\theta$ **do**
      Solve the optimization problem given by the Bellman equation (e.g., Eq. (27)) at $\hat{v}_\theta^{(i)}$ using the predictive means from iteration $\tau - 1$ to obtain $\bar{K}_\theta^{(i),\tau}$ and associated policies.
    end
    Define the training set at iteration τ: $\mathcal{D}_\theta^\tau = \{(\hat{v}_\theta^{(1)}, \bar{K}_\theta^{(1),\tau}), ..., (\hat{v}_\theta^{(J)}, \bar{K}_\theta^{(J),\tau})\}$.
    Fit a GP $\mathcal{G}_\theta$ over $\mathcal{D}_\theta^\tau$ (and, if needed, the corresponding policy functions χ_θ^τ).
  end
  if *BAL score of the best-ranked candidate point satisfies* $\alpha(\cdot;\cdot) < \epsilon_{BAL}$ *(Criterion 1)* **then**
    if *Criteria 2 (global convergence), 3 (local accuracy), and 4 (feasibility) are satisfied* **then**
      Terminate the value function iteration.
    end
    else
      Update $\tau \leftarrow \tau + 1$ and continue.
    end
  end
  else
    Update $\tau \leftarrow \tau + 1$ and continue.
  end
end
Model solution: $\bar{K}^* = \{\bar{K}^\tau_{\theta_{(1)}}, ..., \bar{K}^\tau_{\theta_{(m)}}\}$, $\quad \chi^* = \{\chi^\tau_{\theta_{(1)},1}, \cdots, \chi^\tau_{\theta_{(1)},\kappa}, \cdots, \chi^\tau_{\theta_{(m)},1}, \cdots, \chi^\tau_{\theta_{(m)},\kappa}\}$.

B Machine Learning Glossary

In this Appendix, we provide a short glossary of terms that we use in this paper that are common in the machine learning literature. In addition, we try to link this terminology to the terms commonly used in economics. For a more complete overview, see, for example, Goodfellow et al. (2016), Bishop (2006), and https://developers.google.com/machine-learning/glossary, as well as Igami (2020), who clarifies the connections between certain algorithms to develop artificial intelligence and the econometrics of dynamic structural models.

- **Machine learning:** Mitchell (1997) provides a succinct definition: *A computer program is said to learn from experience E with respect to some class of tasks T and performance measure P, if its performance at tasks in T, as measured by P, improves with experience E.* In simple language, machine learning is a field in which human-made algorithms (such as linear regression or GPR) have the ability to *learn* by themselves and make predictions for unseen data. In the context of our proposed algorithm, E corresponds to the training set $\mathcal{D}$ that contains solutions to the Bellman equation at various locations of the state space at a given iteration step j, whereas the term prediction in our context refers to interpolating the value and policy functions.
- **Model:** A machine learning model is a data structure that stores a representation of a dataset, for example, a GP and its hyperparameters that are used to fit a value function based on a collection of Bellman equations solved at various feasible locations in the state space in a given iteration step of the value function iteration algorithm.
- **Training set:** A set of observations used to generate machine learning models. In our concrete case, the training set contains a collection of solutions to the Bellman operator, solved at a particular point in the state space at each iteration step of the value function procedure.
- **Hyperparameters:** They are higher-level properties of a model, such as how fast it can learn, or the complexity of a model. In the context of GPs, the hyperparameters are, for instance, the characteristic lengthscale and variability defining the SE kernel (cf. Eq. (4)).
- **Training:** Training a model such as a GP means learning, that is, determining good values for all the hyperparameters, for example, via maximizing the likelihood.
- **Supervised machine learning:** Training a model using a *labeled* dataset. A label refers to an *answer* portion of an observation in supervised learning. For example, in our case, we have labels to classify observations into *feasible* and *infeasible* (cf. Section 4.1).
  - **Regression:** Predicting a continuous output. In application, prediction refers to interpolate value or policy functions.
  - **Classification:** Predicting a categorical output, such as *feasible* and *infeasible*.

- **Unsupervised learning:** Training a model to find patterns in an unlabeled dataset. An example of unsupervised machine learning popular in economics is principal component analysis (PCA).
- **Reinforcement learning:** A branch of machine learning that, among other things, is concerned with solving dynamic programming problems. Reinforcement learning is a data-driven approach to solving dynamic programming problems, where the data can be created artificially via simulations or by real-life observations. See Sutton and Barto (2018) for a thorough introduction.
- **Active learning:** A training approach in which the algorithm chooses some of the data it learns from. Active learning is particularly valuable when labeled examples are scarce or expensive to obtain, such as solving constrained optimization problems within every step of the value function iteration. Instead of blindly seeking a diverse range of labeled examples, an active learning algorithm selectively seeks the particular range of examples it needs for learning. In the algorithm we propose in this paper, we use BAL to enrich the training set so that the Bellman equations are solved at the locations of the state space where they improve the GP function approximation the most (cf. Section 3.2).

C Computational bottlenecks

Let N be the number of observations in the training set $\mathcal{D}$ (cf. Section 3.1). The computational cost of standard GPR is dominated by the need to perform a Cholesky decomposition of the $N \times N$ covariance matrix at each iteration step of Alg. 1, which scales as $O(N^3)$ (Rasmussen and Williams, 2005). Thus, using standard implementations of GPs will cause problems when N is in the order of $10,000$. In such cases, one has to resort to fast GP approximations. In our applications, we use GPs based on Blackbox Matrix-Matrix multiplication (BBMM; (Gardner et al., 2018)). BBMM inference uses a modified batched version of the conjugate gradients algorithm to derive all terms for training and inference in a single call. BBMM reduces the asymptotic complexity of exact GP inference from $O(N^3)$ to $O(N^2)$. Moreover, there are other recent fast methods available such as KISS-GP (Wilson and Nickisch, 2015) or Deep Kernel Learning (Wilson et al., 2016) that allow GPs to scale up to millions of observations if one prescribes some structure in the prior covariance kernels. Note that we have implemented and tested those kernels in the Python code accompanying this paper. However, in the context of our models, we achieved the best overall performance with BBMM. For a more detailed discussion, see Murphy (2022, Chapter 18.5), and references therein.

D Parallelization

In order to solve "large" problems in a reasonably short time, we use parallel computation. There is one key location where the value function iteration algorithm described in

Section 4.4 can trivially exploit the availability of parallel computing: the evaluation of the Bellman operator. At every iteration step j of the value function procedure, there are $m \cdot N$ *independent* optimization problems that need to be solved and are all independent from each other. Recall that m is the number of discrete states (resulting, for instance, from the number of persistent, discrete shocks in the model) in the model, and N is the number of observations per discrete state. Consequently, the generation of the training data for the GPs is embarrassingly parallel. We now outline the basic steps in our parallelization using the Message Passing Interface (MPI; see, e.g., Skjellum et al. (1999)). For simplicity, we assume that we have n_{cpu} computational cores available, which we refer to as *workers* or *processes* (that correspond to individual MPI processes) from this point on. At each iteration step j of the value function iteration algorithm, we broadcast the current value function to all processes such that every process can evaluate the Bellman operator independently. The communication cost required to perform this operation is negligibly small. Then, the collection of the N training inputs per discrete state θ (see Alg. 1) becomes embarrassingly parallelizable. In consequence, each worker simply evaluates the Bellman operator at a fraction of the test points, that is, and every worker is assigned with a fractional workload equal to solving $(m \cdot N)/n_{\text{cpu}}$ times. This is where most of the computational time is spent. Subsequently, all the workers gather the distributed data. This operation also has a negligible communication cost. Furthermore, the fitting of the GP hyperparameters is also parallelized across MPI workers, thereby also being moderately accelerated.

E Why use Gaussian processes?

To further illustrate the advantages of GPs and provide a broader perspective, we compare them with other popular global solution methods used in economics, including Chebyshev polynomials (see, e.g., Judd (1998), and references therein) and Adaptive Sparse Grids from the classical numerical analysis toolbox (see, e.g., Brumm et al. (2022), and references therein), and Deep Neural Networks as an alternative approach from the machine learning literature, which have recently piqued the interest of computational economists for solving and estimating dynamic models (see, e.g., Chen et al., 2021, Han et al., 2021, Maliar et al., 2021, Azinovic et al., 2022, Kase et al., 2022, Friedl et al., 2023, Azinovic and Žemlička, 2023, Folini et al., 2024, Fernández-Villaverde et al., 2023, Ebrahim Kahou et al., 2024, Payne et al., 2024, and Valaitis and Villa, 2024).

Chebyshev Polynomials: Chebyshev polynomials are effective for approximating smooth functions and offer exponential convergence rates. They facilitate interpolation and differentiation, which are useful in solving dynamic programs. However, they are less suitable for very high-dimensional problems due to the exponential growth of the number of basis functions with dimensionality. Techniques like sparse grids or low-rank tensor approximations are required to make them feasible in higher dimensions, adding complexity.

Adaptive Sparse Grids: Adaptive sparse grids efficiently handle high-dimensional

problems by focusing computational effort on important regions of the domain. They are grid-based methods and require the construction and management of grids, which can be complex in very high dimensions. Adaptive Sparse Grids offer good interpretability and can adaptively refine the solution where needed.

Neural Networks: Neural networks are highly scalable with respect to data size and are capable of approximating complex nonlinear functions in high-dimensional spaces. However, they lack inherent uncertainty quantification unless combined with Bayesian approaches, which can add computational overhead. Neural Networks require careful tuning of hyperparameters and may suffer from issues like overfitting if not properly regularized. Additionally, they are often considered less interpretable compared to GPs.

Gaussian Processes: In our setting, we deem GPs the more suitable choice for several reasons. First, GPs offer expressiveness that allows us to obtain excellent predictions with considerably fewer observations compared to methods like neural networks. This is particularly advantageous when data is scarce or expensive to obtain. Second, GPs provide an estimate of uncertainty or confidence in the predictions through the predictive variance. While the predictive mean is often used as the best guess of the output (i.e., the interpolation value), the full predictive distribution can be used meaningfully. For example, we can estimate a 95% confidence bound for the predictions, which can be used to measure control performance (cf. Section 3.2). Third, GPs allow us to include prior knowledge of the system behavior by defining priors on the hyperparameters or by constructing a particular structure of the covariance function. This feature enables incorporating domain knowledge into the GP model to improve its accuracy. For more reasons on when to apply GPs instead of neural networks, see Murphy (2022, Chapter 18.1).

We summarize the key properties of these four methods in Tab. 9.

Table 9: Comparison of Methods for High-Dimensional Dynamic Models

Properties	**Gaussian Process Regression**	**Neural Networks**	**Adaptive Sparse Grids**	**Chebyshev Polynomials**
Handles High Dimensionality	✓	✓	✓	×
Nonlinear Function Approximation	✓	✓	✓	✓
Probabilistic Predictions	✓	×	×	×
Scalability with Data Size	×	✓	✓	×
Requires Grid Construction	×	×	✓	×
Adaptive Refinement	✓	×	✓	×
Interpretability	✓	×	✓	✓
Parallelization Potential	✓	✓	✓	✓
Hyperparameter Tuning Needed	Moderate	High	Low	Low
Uncertainty Quantification	✓	×	Limited	×

Below we provide brief descriptions of the properties listed in the table:

- **Handles High Dimensionality:** Ability of the method to effectively manage and solve problems with a large number of state variables without significant loss of accuracy or computational feasibility.[48]

[48] A rule of thumb for specifying "high dimensions"in practical applications is when a problem involves more than about four state variables or dimensions. Beyond this point, many numerical methods become computationally infeasible due to the exponential growth in complexity.

- **Nonlinear Function Approximation:** Capability to approximate complex nonlinear functions, essential in dynamic programming where value and policy functions are often nonlinear.
- **Probabilistic Predictions:** Provides not only point estimates but also quantifies uncertainty in predictions, offering confidence intervals or predictive distributions.
- **Scalability with Data Size:** The method remains computationally efficient as the dataset size increases, allowing practical application to large-scale problems.
- **Requires Grid Construction:** Necessitates the creation of a grid over the state space, which can become computationally intensive in high dimensions due to the curse of dimensionality.
- **Adaptive Refinement:** Ability to refine the solution adaptively in regions where higher accuracy is needed, improving efficiency by focusing computational resources effectively.
- **Interpretability:** The extent to which the method's outputs and internal workings can be easily understood and interpreted by the user.
- **Parallelization Potential:** The method can be parallelized, allowing computations to be distributed across multiple processors or cores to reduce computation time.
- **Hyperparameter Tuning Needed:** The level of effort required to select and tune hyperparameters for optimal performance.
- **Uncertainty Quantification:** Provides measures of uncertainty associated with predictions, valuable for assessing the reliability of the results.

In the table, the symbols indicate:

- $\checkmark$ indicates that the method possesses the property.
- $\times$ indicates that the method does not possess the property.
- "Limited" indicates partial or conditional possession of the property.

In summary, the above comparison should highlight that, while alternative methods have their merits, GPs offer a balance of features that make them particularly suitable for our setting. Their expressiveness allows for excellent predictions with fewer observations, a key property when data is expensive—recall that we need to solve costly constrained optimization problems at many points in the state space. GPs also provide inherent uncertainty quantification and enable the incorporation of prior knowledge through the covariance function. These advantages align with our goal of offering a practical and efficient approach for solving high-dimensional dynamic incentive problems.

F Proof of Lemma 2

To prove Lemma 2, we begin by formulating the finite-horizon version of the dynamic incentive problem (22):

$$
\begin{aligned}
K^0(\hat{v}_0(\cdot), \theta_0) = \max_{c_t(\cdot), w_t(\cdot|\cdot)} & \sum_{t=1}^{T} \beta^{t-1} \sum_{\theta \in \Theta} \pi(\theta|\theta_{t-1})\, v(c_t(\theta), \theta_{t-1}) \\
& \text{subject to} \\
& \hat{v}_t\left(\theta_{t(j)}\right) = \sum_{\theta \in \Theta} \pi\left(\theta|\theta_{(j)}\right) [U(c_t(\theta), \theta) + \beta w_t(\theta|\theta)], \quad \forall j, t < T, \\
& U(c_t(\theta), \theta) + \beta w_t(\theta|\theta) \geq U(c_t(\hat{\theta}), \theta) + \beta w_t(\hat{\theta}|\theta), \quad \forall \theta, \hat{\theta} \in \Theta, t < T, \\
& \hat{v}_{t+1}(\cdot) = w_t(\theta_{t+1}|\cdot), \\
& \hat{v}_T\left(\theta_{T(j)}\right) = \sum_{\theta \in \Theta} \pi\left(\theta|\theta_{(j)}\right) [U(c_T(\theta), \theta)], \quad \forall j \in \{1, \ldots, m\}, \\
& U(c_T(\theta), \theta) \geq U(c_T(\hat{\theta}), \theta), \quad \forall \theta, \hat{\theta} \in \Theta, \\
& c_t(\theta) \in C, \quad \forall \theta \in \Theta, \forall t,
\end{aligned}
\tag{54}
$$

where $\hat{v}_0(\cdot)$ denotes the initial promise, and θ_0 is the starting type. The critical observation here is that the entire problem is solved in a single step by simultaneously accounting for all possible realizations of the shocks. As a result, there is no need to relax the promise-keeping constraints or explicitly determine the feasible set beyond the initial *vector of utility promises*. Starting with any initial feasible promise $\hat{v}_0(\cdot)$ guarantees that the optimal solution will remain feasible for problem (54).

To see that no penalty is required, observe that if we had an infeasible policy $(c_t(\cdot))_t$, there would exist some $s \in \{0, \ldots, T-1\}$ for which at least one of the constraints in (54) is violated, rendering $(c_t(\cdot))_t$ infeasible for (54). Consequently, problem (54) is equivalent to the finite-horizon version of the original problem in (18).[49]

Next, we summarize results from the optimization literature on penalty relaxations, following Ben-Tal and Nemirovski (2023, Section 9.1). Consider the optimization problem

$$
\min_x f(x) \quad \text{s.t.} \quad h_i(x) = 0, \quad g_j(x) \leq 0, \quad i \in \{1, \ldots, k\},\ j \in \{1, \ldots, l\}, \tag{55}
$$

where f, h_i, and g_j are continuous functions. The penalty approach reformulates this problem as an unconstrained optimization problem by incorporating the constraint vio-

[49] This equivalence holds only under finite time and a finite number of shocks, which ensures that every possible reporting sequence can be accounted for.

lations into the objective function. This yields the penalty reformulation

$$\min_x f(x) + \frac{1}{2}\rho\left(\sum_{i=1}^{k}(h_i(x))^2 + \sum_{j=1}^{l}(\max\{0, g_j(x)\})^2\right), \tag{56}$$

where $\rho > 0$ is the penalty parameter.

Theorem 1 (Ben-Tal and Nemirovski (2023), Th. 9.2.1)**.** *Let the objective f in problem (55) possess bounded level sets:*

$$\{x \mid f(x) \leq \ell\} \text{ is bounded for all } \ell,$$

and let problem (55) be feasible. Then, for any positive ρ, the set of global minimizers $X^*(\rho)$ of the penalized objective f_ρ is nonempty. Moreover, if X^* is the set of globally optimal solutions to (55), then for large ρ, $X^*(\rho)$ is "close" to X^*: for any $\epsilon > 0$, there exists $\rho = \rho(\epsilon)$ such that $X^*(\rho)$, for all $\rho \geq \rho(\epsilon)$, is contained within the ϵ-neighborhood

$$X^*_\epsilon = \{x \mid \exists x^* \in X^* : \|x - x^*\| < \epsilon\}$$

of the optimal set of (55).

We are now prepared to prove Lemma 2.

Proof. The main requirement of the theorem, aside from the continuity of the functions, is that the level sets of the objective are bounded. Initially, the level sets of the objective in (54) are unbounded, so we equivalently reformulate the objective as

$$\sum_{t=0}^{T-1}\beta^t \sum_{\theta\in\Theta}\pi(\theta|\theta_{t-1})\, v(c_t(\theta), \theta_{t-1}) + \min\{0, \log(\overline{c} + 1 - c_t(\theta_t))\}, \tag{57}$$

where $\overline{c}$ is an upper bound on consumption, and $\min\{0, \log(\overline{c} + 1 - c_t(\theta_t))\}$ is a barrier function that remains zero for any $c_t(\cdot) \in C$.

This reformulation has the same solution as the original problem, since the added barrier only becomes active outside the feasible set. Thus, problem (57) now has bounded level sets. Furthermore, note that all constraints are defined by a finite number of continuous functions, which implies that the feasible set is closed. By Ben-Tal and Nemirovski (2023, Th. 9.2.1), the penalty relaxation (56) with a quadratic penalty function possesses the desired convergence property.

Finally, observe that we only relaxed the promise-keeping constraints in (27) rather than all constraints. However, the argument in Ben-Tal and Nemirovski (2023, Th. 9.2.1) extends to this partially relaxed problem as well. □

G Alternative Sampling Strategy

In state spaces with four or more dimensions, the feasible region occupies a very small fraction of $\mathcal{B}$. Consequently, uniformly sampling from $\mathcal{B}$ becomes highly inefficient because only a minor fraction of the points are in the feasible set, necessitating a prohibitively large initial sample. To address this limitation, we propose an alternative sampling strategy that locates initial samples closer to the region of interest.

We begin by assuming that future utility promises equal current promises (i.e., a steady-state condition) and then maximize a random hyperplane subject to the model constraints, as specified by:

$$
\begin{aligned}
&\max_{c(\cdot),\hat{v}(\cdot)} \sum_{k=1}^{m} a_k \hat{v}\left(\theta_{(k)}\right) \\
&\text{subject to} \\
&\hat{v}\left(\theta_{(k)}\right) + \xi_k = \sum_{j=1}^{m} \pi\left(\theta_{(j)}|\theta_{(k)}\right)\left[U(c(\theta_{(k)}), \theta_{(k)}) + \beta \hat{v}\left(\theta_{(j)}\right)\right], \\
&U(c(\theta_{(i)}), \theta_{(i)}) + \beta \hat{v}\left(\theta_{(i)}\right) \geq U(c(\theta_{(k)}), \theta_{(i)}) + \beta \hat{v}\left(\theta_{(i)}\right), \\
&c(\theta_{(k)}) \in C, \quad \hat{v}\left(\theta_{(k)}\right) \in \mathcal{B} = \prod_{j=1}^{m}\left[U(\underline{c}, \theta_{(j)})/(1-\beta), U(\bar{c}, \theta_{(j)})/(1-\beta)\right], \\
&\forall k \in \{1, \ldots, m\}, \quad \forall i \in \{1, \ldots, m\},
\end{aligned}
\tag{58}
$$

where a_k are random coefficients drawn from the unit sphere. The optimal solution to this problem lies on the boundary of the set of stationary points. Any point in this set is feasible, as the principal can always maintain the current vector of promises.

We solve this problem multiple times with different a_k values, yielding several boundary points depending on the model. For example, in the 10-type model analyzed in Section 4.5.2, we solve 400 instances of this problem, which produces approximately 80 distinct boundary points. However, since these points lie only on the boundary, a random sample of interior points is needed to refine the initial guess. Thus, we construct our initial samples by taking a uniform random selection (e.g., 32 samples, as in Section 4.5.2) from the convex hull of these solutions and adding a statistical noise term. Depending on the magnitude of the noise, these points will either be close to or within the feasible set, making them suitable starting values for initializing the value function iteration. For further implementation details, refer to the supplementary code.

References

Abraham, A. and N. Pavoni (2008): "Efficient allocations with moral hazard and hidden borrowing and lending: A recursive formulation," *Review of Economic Dynamics*, 11, 781 – 803.

Abreu, D., B. Brooks, and Y. Sannikov (2020): "Algorithms for Stochastic Games With Perfect Monitoring," *Econometrica*, 88, 1661–1695.

Abreu, D., D. Pearce, and E. Stacchetti (1986): "Optimal cartel equilibria with imperfect monitoring," *Journal of Economic Theory*, 39, 251 – 269.

——— (1990): "Toward a Theory of Discounted Repeated Games with Imperfect Monitoring," *Econometrica*, 58, 1041–1063.

Abreu, D. and Y. Sannikov (2014): "An algorithm for two-player repeated games with perfect monitoring," *Theoretical Economics*, 9, 313–338.

Akerlof, G. A. (1970): "The Market for "Lemons": Quality Uncertainty and the Market Mechanism," *The Quarterly Journal of Economics*, 84, 488–500, publisher: Oxford University Press.

Albanesi, S. and C. Sleet (2006): "Dynamic Optimal Taxation with Private Information," *The Review of Economic Studies*, 73, 1–30.

Azinovic, M., L. Gaegauf, and S. Scheidegger (2022): "Deep equilibrium nets," *International Economic Review*, 63, 1471–1525.

Azinovic, M. and J. Žemlička (2023): "Economics-inspired neural networks with stabilizing homotopies," *arXiv preprint arXiv:2303.14802*.

Balandat, M., B. Karrer, D. R. Jiang, S. Daulton, B. Letham, A. G. Wilson, and E. Bakshy (2020): "BoTorch: A Framework for Efficient Monte-Carlo Bayesian Optimization," in *Advances in Neural Information Processing Systems 33*.

Battaglini, M. and R. Lamba (2019): "Optimal dynamic contracting: The first-order approach and beyond," *Theoretical Economics*, 14, 1435–1482, _eprint: https://onlinelibrary.wiley.com/doi/pdf/10.3982/TE2355.

Bellman, R. (1961): *Adaptive Control Processes: A Guided Tour*, 'Rand Corporation. Research studies, Princeton University Press.

Ben-Tal, A. and A. Nemirovski (2023): "Optimization III: Convex Analysis, Nonlinear Programming Theory, Nonlinear Programming Algorithms," Https://www2.isye.gatech.edu/ nemirovs/OPTIIILN2023Spring.pdf.

Berkenkamp, F., A. P. Schoellig, and A. Krause (2016): "Safe controller optimization for quadrotors with Gaussian processes," in *2016 IEEE International Conference on Robotics and Automation (ICRA)*, IEEE, 491–496.

Bertsekas, D. P. (2000): *Dynamic Programming and Optimal Control*, Athena Scientific, 2nd ed.

Bilionis, I., N. Zabaras, B. Konomi, and G. Lin (2013): "Multi-output separable Gaussian process: Towards an efficient, fully Bayesian paradigm for uncertainty quantification," *Journal of Computational Physics*, 241, 212–239.

Bishop, C. M. (2006): *Pattern Recognition and Machine Learning (Information Science and Statistics)*, Berlin, Heidelberg: Springer-Verlag.

Bloedel, A. W., R. Krishna, and O. Leukhina (2024): "Insurance and Inequality with Persistent Private Information," Working Paper 2018-020, Federal Reserve Bank of St. Louis.

Bloedel, A. W., R. V. Krishna, and B. H. Strulovici (2020): "Persistent private information revisited," *Available at SSRN 4778185*.

Broer, T., M. Kapicka, and P. Klein (2017): "Consumption risk sharing with private information and limited enforcement," *Review of Economic Dynamics*, 23, 170 – 190.

Brumm, J., C. Krause, A. Schaab, and S. Scheidegger (2022): "Sparse grids for dynamic economic models," in *Oxford Research Encyclopedia of Economics and Finance*, Oxford University Press.

Brumm, J. and S. Scheidegger (2017): "Using Adaptive Sparse Grids to Solve High-Dimensional Dynamic Models," *Econometrica*, 85, 1575–1612.

Busoniu, L., R. Babuska, B. D. Schutter, and D. Ernst (2010): *Reinforcement Learning and Dynamic Programming Using Function Approximators*, USA: CRC Press, Inc., 1st ed.

Cai, Y. and K. L. Judd (2014): "Advances in numerical dynamic programming and new applications," *Handbook of computational economics*, 3.

Chaloner, K. and I. Verdinelli (1995): "Bayesian Experimental Design: A Review," *Statist. Sci.*, 10, 273–304.

Chen, H., A. Didisheim, and S. Scheidegger (2021): "Deep surrogates for finance: With an application to option pricing," Available at SSRN 3782722.

Cole, H. L. and N. R. Kocherlakota (2001): "Efficient Allocations with Hidden Income and Hidden Storage," *The Review of Economic Studies*, 68, 523–542.

Deisenroth, M. P., C. E. Rasmussen, and J. Peters (2009): "Gaussian process dynamic programming," *Neurocomputing*, 72, 1508–1524.

DeMarzo, P. M. and Y. Sannikov (2006): "Optimal Security Design and Dynamic Capital Structure in a Continuous-Time Agency Model," *The Journal of Finance*, 61, 2681–2724.

Di Fiore, F., M. Nardelli, and L. Mainini (2024): "Active learning and bayesian optimization: a unified perspective to learn with a goal," *Archives of Computational Methods in Engineering*, 1–29.

Doepke, M. and R. M. Townsend (2006): "Dynamic mechanism design with hidden income and hidden actions," *Journal of Economic Theory*, 126, 235 – 285.

Ebrahim Kahou, M., J. Fernández-Villaverde, S. Gomez-Cardona, J. Perla, and J. Rosa (2024): "Spooky boundaries at a distance: Inductive bias, dynamic models, and behavioral macro," Working Paper 32850, National Bureau of Economic Research.

Eftekhari, A. and S. Scheidegger (2022): "High-Dimensional Dynamic Stochastic Model Representation," *SIAM Journal on Scientific Computing*, 44, C210–C236.

Engel, Y., S. Mannor, and R. Meir (2003): "Bayes Meets Bellman: The Gaussian Process Approach to Temporal Difference Learning," in *Proceedings of the 20th International Conference on Machine Learning*, AAAI Press, 154–161.

Feldman, N. E. and J. Slemrod (2007): "Estimating tax noncompliance with evidence from unaudited tax returns," *The Economic Journal*, 117, 327–352.

Fernandes, A. and C. Phelan (2000): "A Recursive Formulation for Repeated Agency with History Dependence," *Journal of Economic Theory*, 91, 223 – 247.

Fernández-Villaverde, J., G. Gordon, P. Guerrón-Quintana, and J. F. Rubio-Ramirez (2015): "Nonlinear adventures at the zero lower bound," *Journal of Economic Dynamics and Control*, 57, 182–204.

Fernández-Villaverde, J., S. Hurtado, and G. Nuño (2023): "Financial frictions and the wealth distribution," *Econometrica*, 91, 869–901.

Folini, D., A. Friedl, F. Kübler, and S. Scheidegger (2024): "The climate in climate economics," *Review of Economic Studies*, rdae011.

Friedl, A., F. Kübler, S. Scheidegger, and T. Usui (2023): "Deep uncertainty quantification: With an application to integrated assessment models," Working paper, University of Lausanne.

Gaegauf, L., S. Scheidegger, and F. Trojani (2023): "A Comprehensive Machine Learning Framework for Dynamic Portfolio Choice With Transaction Costs," *Available at SSRN 4543794*.

Gardner, J., G. Pleiss, K. Q. Weinberger, D. Bindel, and A. G. Wilson (2018): "Gpytorch: Blackbox matrix-matrix gaussian process inference with gpu acceleration," *Advances in neural information processing systems*, 31.

Genz, A. (1984): "Testing Multidimensional Integration Routines," in *Proc. Of International Conference on Tools, Methods and Languages for Scientific and Engineering Computation*, New York, NY, USA: Elsevier North-Holland, Inc., 81–94.

Golosov, M., A. Tsyvinski, and N. Werquin (2016): "Recursive Contracts and Endogenously Incomplete Markets," *Handbook of Macroeconomics*, 2, 725–841.

Goodfellow, I., Y. Bengio, and A. Courville (2016): *Deep Learning*, MIT Press, http://www.deeplearningbook.org.

Green, E. J. (1987): "Lending and the smoothing of uninsurable income," *Contractual arrangements for intertemporal trade*, 1, 3–25.

Haan, W. J. D., K. L. Judd, and M. Juillard (2011): "Computational suite of models with heterogeneous agents II: Multi-country real business cycle models," *Journal of Economic Dynamics and Control*, 35, 175 – 177, computational Suite of Models with Heterogeneous Agents II: Multi-Country Real Business Cycle Models.

Halton, J. H. (1964): "Algorithm 247: Radical-inverse Quasi-random Point Sequence," *Commun. ACM*, 7, 701–702.

Han, J., Y. Yang, and W. E (2021): "DeepHAM: A global solution method for heterogeneous agent models with aggregate shocks," *arXiv preprint arXiv:2112.14377*.

He, Z., B. Wei, J. Yu, and F. Gao (2017): "Optimal Long-Term Contracting with Learning," *The Review of Financial Studies*, 30, 2006–2065.

Igami, M. (2020): "Artificial intelligence as structural estimation: Deep Blue, Bonanza, and AlphaGo," *The Econometrics Journal*, 23, S1–S24.

Judd, K., S. Yeltekin, and J. Conklin (2003): "Computing Supergame Equilibria," *Econometrica*, 71, 1239–1254.

Judd, K. L. (1998): *Numerical methods in economics*, The MIT press.

Judd, K. L., L. Maliar, S. Maliar, and R. Valero (2014): "Smolyak method for solving dynamic economic models: Lagrange interpolation, anisotropic grid and adaptive domain," *Journal of Economic Dynamics and Control*, 44, 92–123.

Kapička, M. (2013): "Efficient Allocations in Dynamic Private Information Economies with Persistent Shocks: A First-Order Approach," *The Review of Economic Studies*, 80, 1027–1054.

Kase, H., L. Melosi, and M. Rottner (2022): "Estimating nonlinear heterogeneous agents models with neural networks," Discussion Paper 17391, CEPR.

Keane, M. P. and K. I. Wolpin (1994): "The Solution and Estimation of Discrete Choice Dynamic Programming Models by Simulation and Interpolation: Monte Carlo Evidence," *The Review of Economics and Statistics*, 76, 648–672.

Kocherlakota, N. R. (2005): "Zero Expected Wealth Taxes: A Mirrlees Approach to Dynamic Optimal Taxation," *Econometrica*, 73, 1587–1621.

——— (2010): *The new dynamic public finance*, Princeton University Press.

Kotlikoff, L., F. Kubler, A. Polbin, and S. Scheidegger (2021): "Pareto-improving carbon-risk taxation," *Economic Policy*, 36, 551–589.

Krause, A. and C. Guestrin (2007): "Nonmyopic Active Learning of Gaussian Processes: An Exploration-Exploitation Approach," in *Proceedings of the 24th International Conference on Machine Learning*, New York, NY, USA: Association for Computing Machinery, ICML '07, 449–456.

Krause, A., A. Singh, and C. Guestrin (2008): "Near-Optimal Sensor Placements in Gaussian Processes: Theory, Efficient Algorithms and Empirical Studies," *J. Mach. Learn. Res.*, 9, 235–284.

Krueger, D. and F. Kubler (2004): "Computing equilibrium in OLG models with stochastic production," *Journal of Economic Dynamics and Control*, 28, 1411 – 1436.

Lambert, R. A. (1983): "Long-term Contracts and Moral Hazard," *Bell Journal of Economics*, 14, 441–452.

Ljungqvist, L. and T. Sargent (2000): *Recursive macroeconomic theory*, Mit Press.

Luenberger, D. G. and Y. Ye (2008): *Linear and Nonlinear Programming*, International Series in Operations Research & Management Science, Springer US, 3 ed.

MacKay, D. J. C. (1992): "Information-Based Objective Functions for Active Data Selection," *Neural Computation*, 4, 590–604.

Mailath, G. J., I. Obara, and T. Sekiguchi (2002): "The Maximum Efficient Equilibrium Payoff in the Repeated Prisoners' Dilemma," *Games and Economic Behavior*, 40, 99 – 122.

Makarova, A., H. Shen, V. Perrone, A. Klein, J. B. Faddoul, A. Krause, M. Seeger, and C. Archambeau (2022): "Automatic Termination for Hyperparameter Optimization," Tech. rep.

Maliar, L. and S. Maliar (2014): "Numerical methods for large-scale dynamic economic models," in *Handbook of computational economics*, Elsevier, vol. 3, 325–477.

Maliar, L., S. Maliar, and P. Winant (2021): "Deep learning for solving dynamic economic models." *Journal of Monetary Economics*, 122, 76–101.

Marcet, A. and R. Marimon (2019): "Recursive Contracts," *Econometrica*, 87, 1589–1631.

Mathevet, L., D. Pearce, and E. Stacchetti (2022): "Reputation for A Degree of Honesty," Tech. rep., Working Paper.

Meghir, C. and L. Pistaferri (2004): "Income Variance Dynamics and Heterogeneity," *Econometrica*, 72, 1–32.

Mele, A. (2014): "Repeated moral hazard and recursive Lagrangeans," *Journal of Economic Dynamics and Control*, 42, 69 – 85.

Micchelli, C. A., Y. Xu, and H. Zhang (2006): "Universal Kernels," *J. Mach. Learn. Res.*, 7, 2651–2667.

Mitchell, T. (1997): *Machine Learning*, McGraw-Hill International Editions, McGraw-Hill.

Murphy, K. P. (2012): *Machine Learning: A Probabilistic Perspective*, The MIT Press.

——— (2022): *Probabilistic Machine Learning: Advanced Topics*, MIT Press.

Niederreiter, H. (1992): *Random Number Generation and quasi-Monte Carlo Methods*, Philadelphia, PA, USA: Society for Industrial and Applied Mathematics.

Oberkampf, W. L. and C. J. Roy (2010): *Verification and Validation in Scientific Computing*, Cambridge University Press.

Park, J. and I. W. Sandberg (1991): "Universal Approximation Using Radial-Basis-Function Networks," *Neural Computation*, 3, 246–257.

Pavan, A., I. Segal, and J. Toikka (2014): "Dynamic mechanism design: A myersonian approach," *Econometrica*, 82, 601–653.

Pavoni, N., C. Sleet, and M. Messner (2017): "The Dual Approach to Recursive Optimization: Theory and Examples," *Forthcoming, Econometrica*.

Payne, J., A. Rebei, and Y. Yang (2024): "Deep learning for search and matching models," Available at SSRN 4768566.

Poggio, T. and F. Girosi (1990): "Networks for approximation and learning," *Proceedings of the IEEE*, 78, 1481–1497.

Rasmussen, C. E. and H. Nickisch (2010): "Gaussian Processes for Machine Learning (GPML) Toolbox," *Journal of Machine Learning Research*, 11, 3011–3015.

Rasmussen, C. E. and C. K. I. Williams (2005): *Gaussian Processes for Machine Learning (Adaptive Computation and Machine Learning)*, The MIT Press.

Rogerson, W. P. (1985): "Repeated Moral Hazard," *Econometrica*, 53, 69–76.

Rust, J. P. (1996): "Numerical dynamic programming in economics," in *Handbook of Computational Economics*, ed. by H. M. Amman, D. A. Kendrick, and J. Rust, Elsevier, vol. 1, chap. 14, 619–729, 1 ed.

Sandroni, A. and F. Squintani (2007): "Overconfidence, Insurance, and Paternalism," 97, 1994–2004.

Sannikov, Y. (2008): "A Continuous- Time Version of the Principal: Agent Problem," *The Review of Economic Studies*, 75, 957–984.

——— (2022): "Breaking the Curse of Dimensionality," Tech. rep.

Scheidegger, S. and I. Bilionis (2019): "Machine learning for high-dimensional dynamic stochastic economies," *Journal of Computational Science*, 33, 68 – 82.

Skjellum, A., W. Gropp, and E. Lusk (1999): *Using MPI*, MIT Press.

Sleet, C. and S. Yeltekin (2016): "On the Computation of Value Correspondences for Dynamic Games," *Dynamic Games and Applications*, 6, 174–186.

Spear, S. E. and S. Srivastava (1987): "On Repeated Moral Hazard with Discounting," *The Review of Economic Studies*, 54, 599–617.

Stokey, N., R. Lucas, and E. Prescott (1989): "Recursive methods in economic dynamics," *Cambridge MA*.

Storesletten, K., C. Telmer, and A. Yaron (2004): "Consumption and risk sharing over the life cycle," *Journal of Monetary Economics*, 51, 609 – 633.

Sutton, R. S. and A. G. Barto (2018): *Reinforcement learning: An introduction*, MIT press.

Thomas, J. and T. Worrall (1990): "Income fluctuation and asymmetric information: An example of a repeated principal-agent problem," *Journal of Economic Theory*, 51, 367–390.

Valaitis, V. and A. T. Villa (2024): "A machine learning projection method for macro-finance models," *Quantitative Economics*, 15, 145–173.

Waechter, A. and L. T. Biegler (2006): "On the implementation of an interior-point filter line-search algorithm for large-scale nonlinear programming," *Math. Program.*, 106, 25–57.

Wagberg, J., D. Zachariah, T. Schon, and P. Stoica (2017): "Prediction performance after learning in Gaussian process regression," in *Artificial Intelligence and Statistics*, PMLR, 1264–1272.

Werning, I. (2002): "Optimal Unemployment Insurance with Unobservable Savings," .

Williams, N. (2009): "On Dynamic Principal-Agent Problems in Continuous Time," .

——— (2011): "Persistent Private Information," *Econometrica*, 79, 1233–1275.

Wilson, A. and H. Nickisch (2015): "Kernel interpolation for scalable structured Gaussian processes (KISS-GP)," in *International conference on machine learning*, PMLR, 1775–1784.

Wilson, A. G., Z. Hu, R. Salakhutdinov, and E. P. Xing (2016): "Deep kernel learning," in *Artificial intelligence and statistics*, PMLR, 370–378.

Wu, J., M. Poloczek, A. G. Wilson, and P. Frazier (2017): "Bayesian optimization with gradients," *Advances in neural information processing systems*, 30.

Wu, Y., H. Wang, B. Zhang, and K.-L. Du (2012): "Using Radial Basis Function Networks for Function Approximation and Classification," *ISRN Applied Mathematics*.

Yeltekin, S., Y. Cai, and K. L. Judd (2017): "Computing Equilibria of Dynamic Games," *Operations Research*, 65, 337–356.

Zhang, Y. (2009): "Dynamic contracting with persistent shocks," *Journal of Economic Theory*, 144, 635–675.